Trend-based, Holistic Observation, Manipulation & Analysis Software

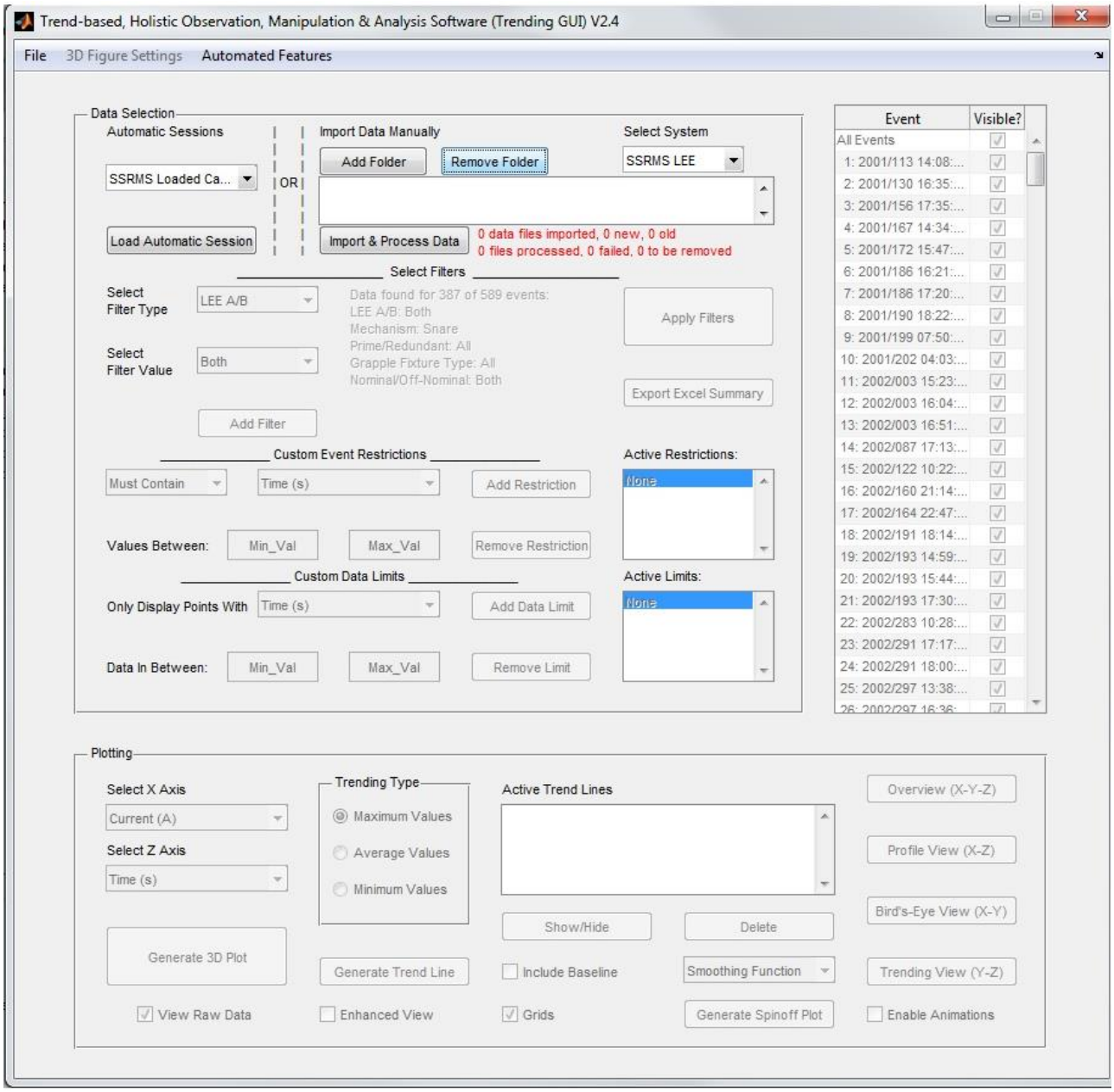


Table of Contents

List of Tables

List of Figures

1. Introduction

The Trending GUI is a tool that quickly and easily generates plots of a repeating type of event (Ex. all loaded LEE latching ops), to allow the user to filter the data down to a desired subset and modify the view settings from a single interface. The foundation of this data visualization tool is the 3D plot, which shows the progression over time (plotted in the Y-axis) of specified profiles (plotted in the X- and Z-axes). Showing a more complete set of data provides context for any trend lines or spinoff plots generated, and can make it easier to spot areas of interest than if you were to jump straight to the 2D trend lines. It also operates on the principle that a human is much better at spotting peculiar or off-nominal behavior than an automated program. The GUI was written using MATLAB 2013b.

2. Quick Start Guide

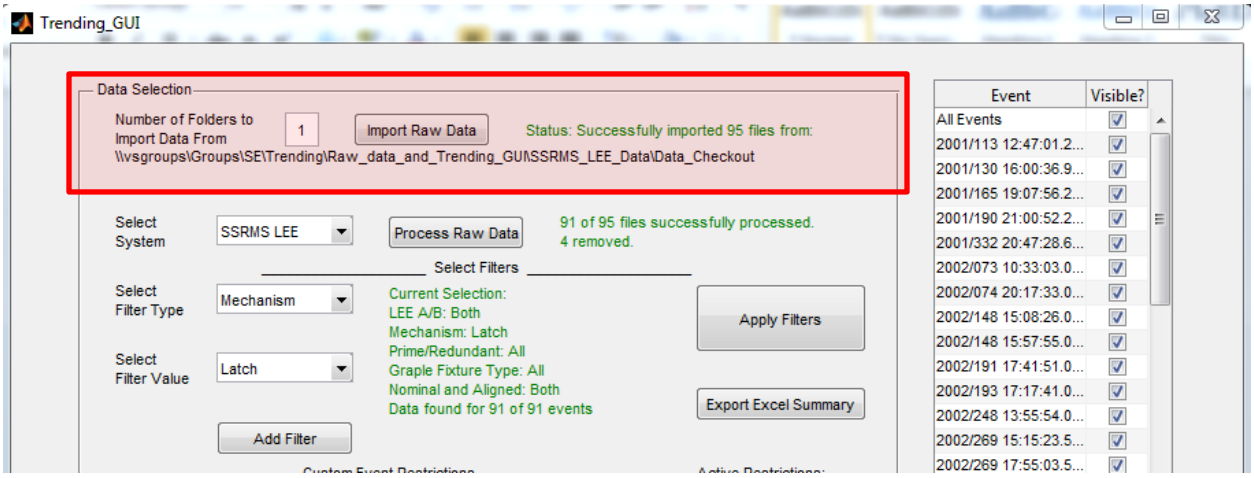
Start the GUI by running the **Trending_GUI.m** script. Opening **Trending_GUI.fig** alone will not initialize the GUI functionality properly.

To generate plots, there are four steps that need to be taken:

1) Import the raw data from text files of the same format as those text files extracted from Mass Data using “save as”.
2) Put the data through a processing function specific to the mechanism being analyzed.
3) Select additional filters to be applied to the data.
4) Generate the 3D plot with specified parameters for the X- and Z- axes.

Once a 3D plot has been generated, trend lines can be added and spinoff plots can be generated. Note that importing data is the most computationally time-consuming step, and steps 1) and 2) can be bypassed with the save/load functionality described in section 3. Menu Bar Functionality.

Step 1: Importing data



In this step, data is automatically imported into MATLAB variables using the *importdata()* function. The text data files (.txt) must be tab-delimited, with the first row containing the column headings, and the first column containing the timestamps of the operation which is the format of files exported by Mass Data. The GUI will automatically try to import every .txt file from the folders selected. Multiple folders may be selected by changing the value in the box to the left of the Import Raw Data button. Databases of raw data text files have already been created in the following directories:

\\vsgroups\Groups\SE\Trending\Raw_data_and_Trending_GUI\SPDM_OTCM_Data

\\vsgroups\Groups\SE\Trending\Raw_data_and_Trending_GUI\SSRMS_LEE_Data

Step 2: Processing data

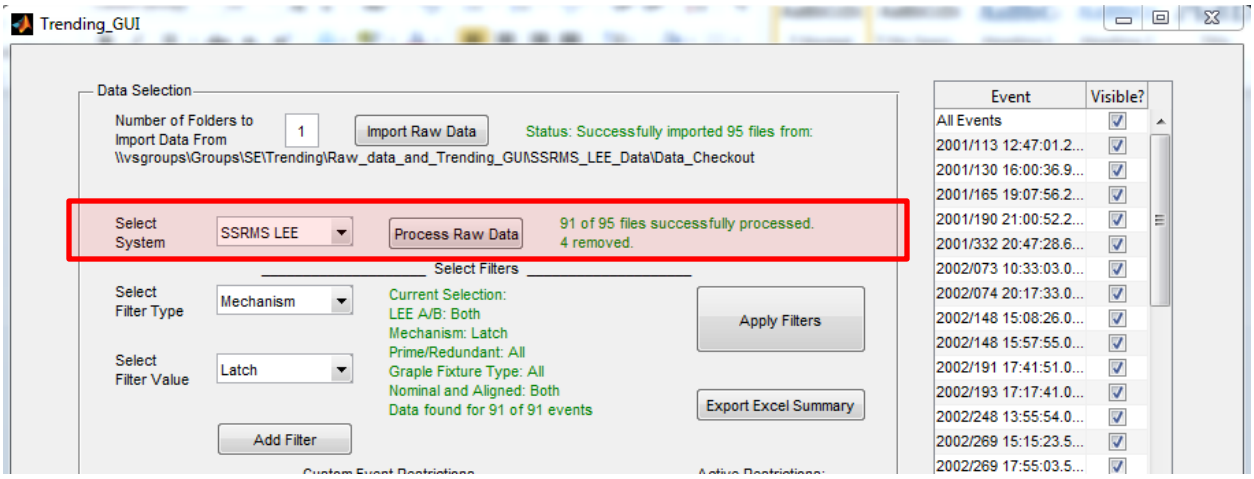


Figure 1: Process Raw Data

In Step 2, the raw data is put through a function (described in Appendix A: Data Processing Function Details) that removes bad data points and creates all of the filter information required by the GUI. This includes the names of the filters and their options, as well as information on which data points satisfy those filters. At the time of creation of this user manual (March 2015), there were two processing

functions implemented: one for the SSRMS LEE mechanisms and one for the SPDM OTCM mechanisms. Instructions on modifying or creating new processing functions can be found in Appendix B: Adding a New Data Processing Function.

Step 3: Filtering data

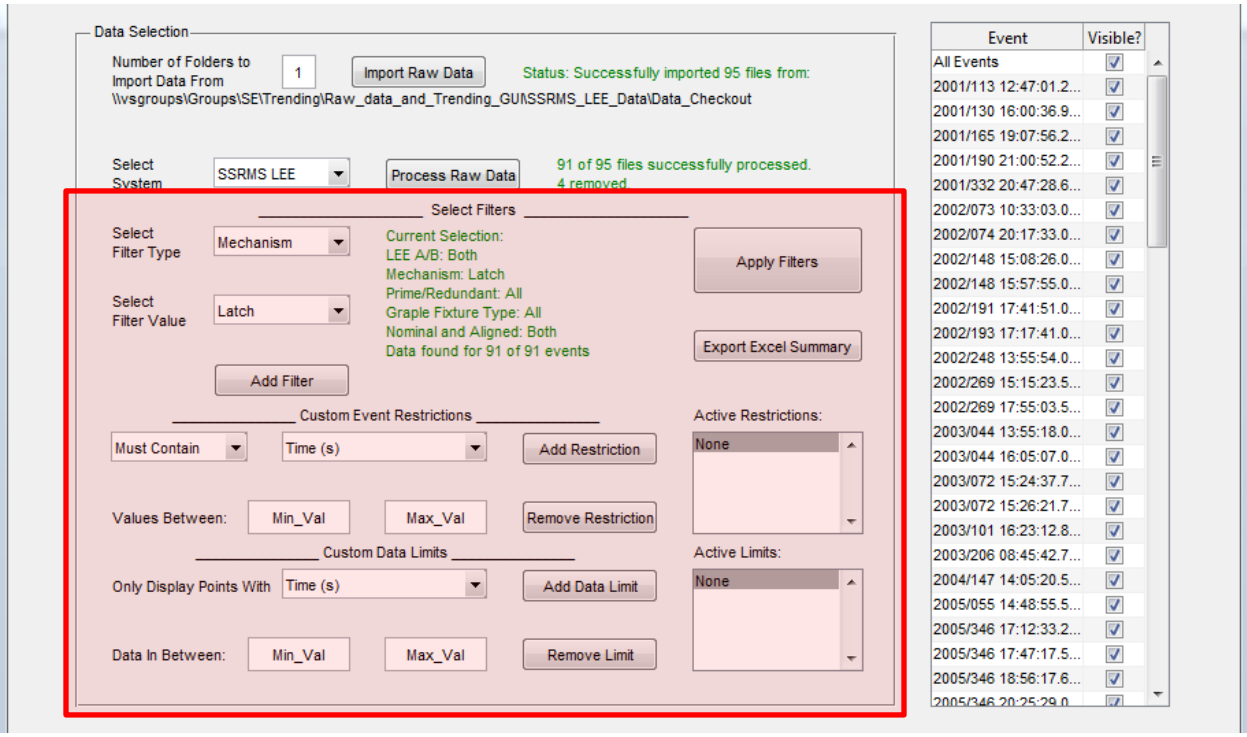


Figure 2: Filtering Data

Step 3 allows the user to customize the filtered data displayed in the 3D plots. There are three ways to apply data filters:

1) **Automatically generated filters**. These filters come directly from the processing function and have predetermined data points that satisfy each of them. To modify these you must:
    a) Select the filter type.
    b) Select the filter value.
    c) Confirm your selection with the **Add Filter** button
    d) Activate selected filters with the **Apply Filters** button. Multiple filters can be added before activating them.

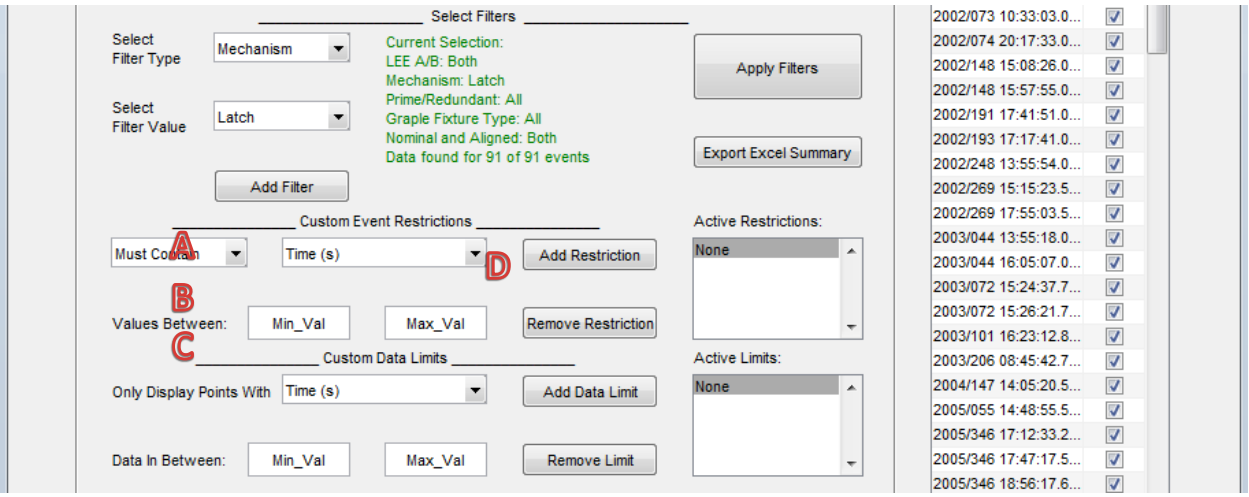


Figure 3: Custom Filters

2) **Adding Event Restrictions.** These filters are applied before **Data Limit** filters, and will remove any event that has a single data point that does not satisfy the selection. To create one:
   a) Select whether you want to make sure each event has/does not have a data point.
   b) Select which parameter to check for the restriction.
   c) Specify the min and max value that the parameter must have/must not have a data point. If either of these fields does not have a numeric value (is blank or only contains text), only one end of the range will be checked.
   d) Add the restriction.
   e) Activate the filters with the **Apply Filters** button.

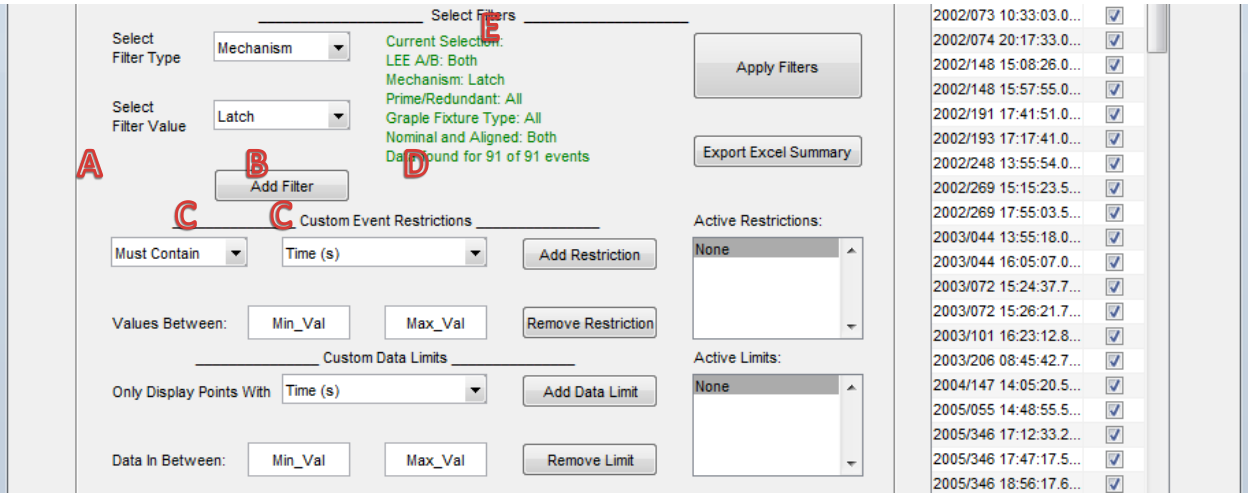


Figure 4: Event Restriction Filters

3) **Adding Data Limits.** These filters are applied after **Event Restrictions** filters. To avoid unexpected results, remove all data limits before applying event restrictions. The Data limits will remove all data points that are not within the specified limits along with associated data points from other parameters (Ex. if at time t = 5, the current does not satisfy the limits imposed, the motor rate and position data at t = 5 will be removed, regardless of whether or not current is being plotted). Data limits will also set axes limits and can be useful even when raw data visibility has been turned off. Similar to event restrictions, to add a limit:
    a) Select which parameter you wish to limit.
    b) Select the desired range for that parameter (only one limit needs to be set).
    c) Add the data limit
    d) Activate the filters with the **Apply Filters** button.

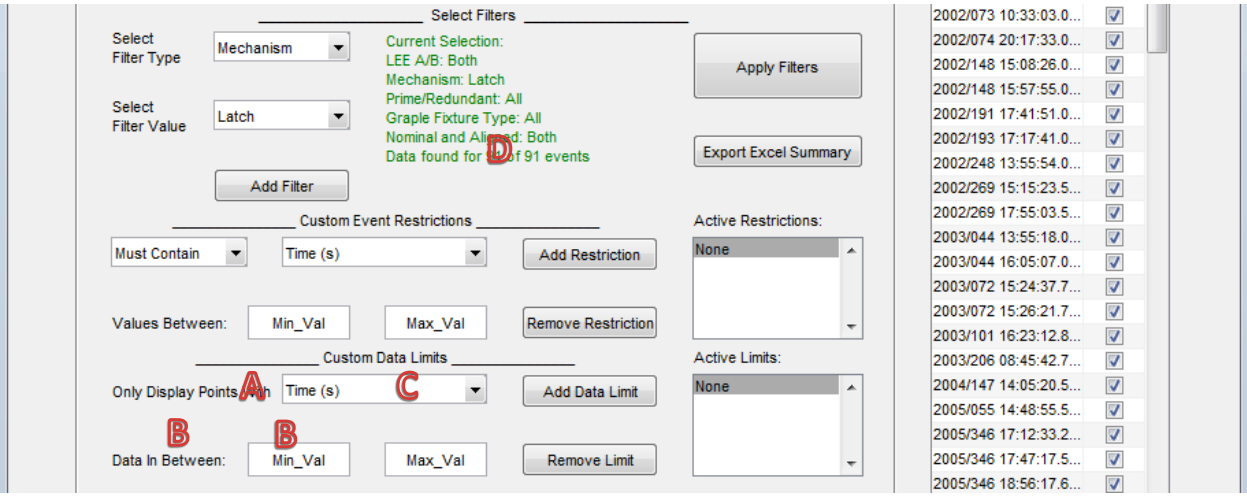


Figure 5: Data Limit Filters

4) **Removing Restrictions/Limits.** To remove restrictions/limits:
    a) Select the restriction/limit.
    b) Remove the restriction/limit.
    c) Activate changes with the **Apply Filters** button.

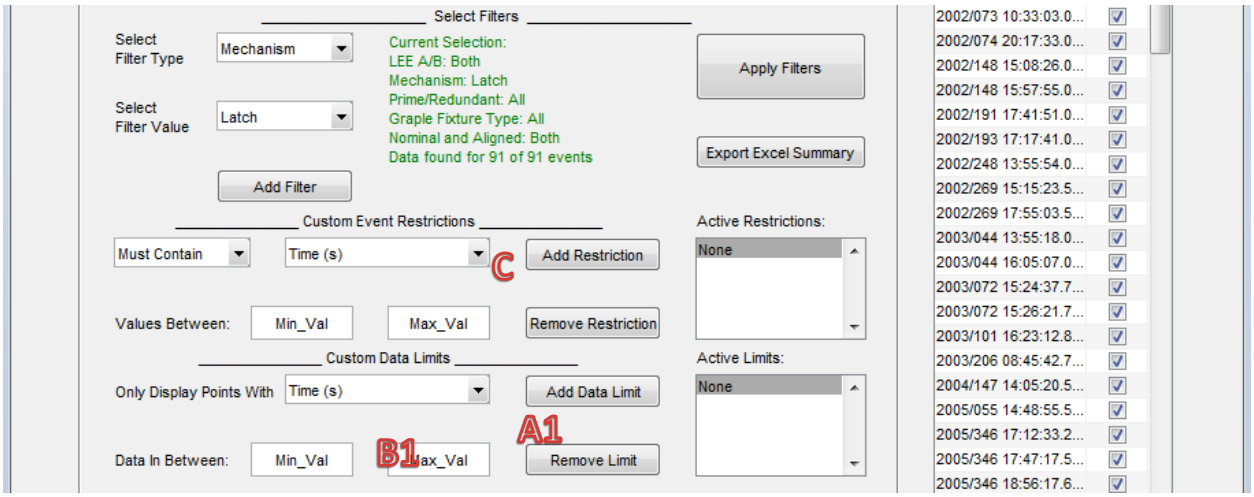


Figure 6: Removing Restrictions/Limits

5) **Manually Filtering Events.** Any event can manually be omitted by deselecting it in the event list on the right hand side. Note that the event list is automatically reset when the **Apply Filters** button is hit and the new event dates do not match the old event dates. To quickly select multiple dates, highlight the Events and right click to select/deselect the highlighted group.

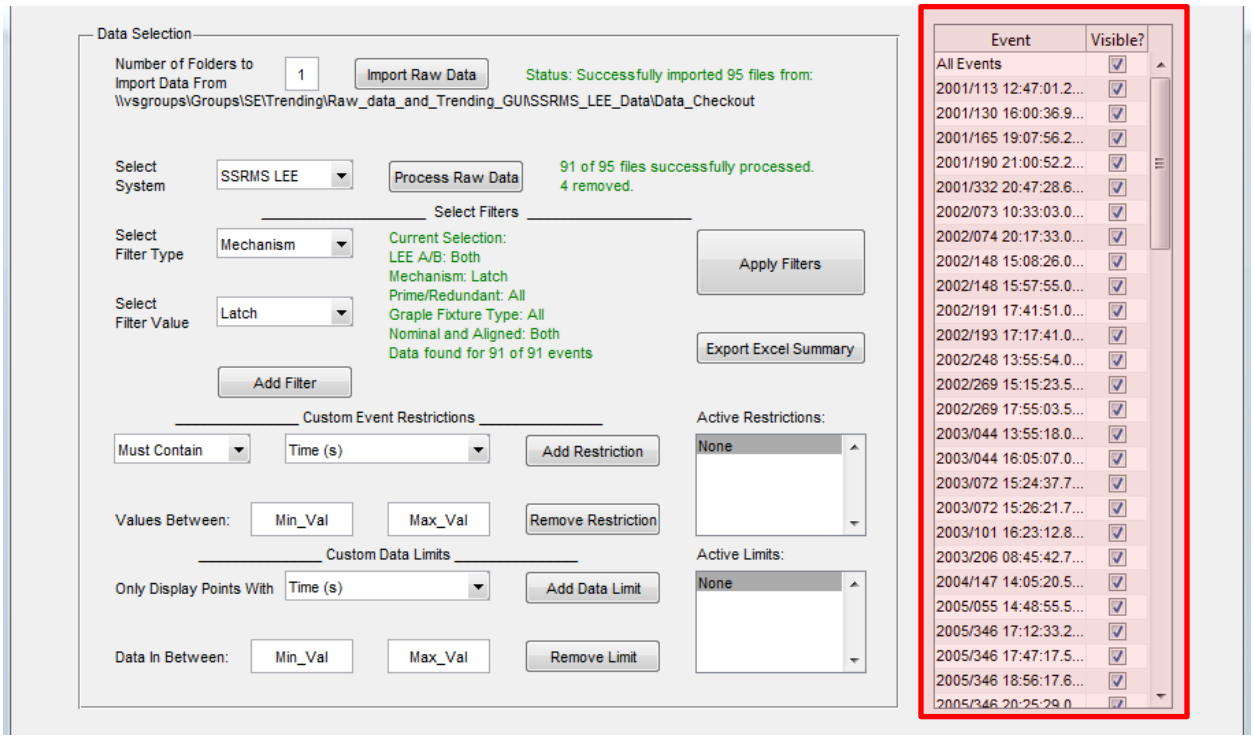


Figure 7: Manually Filtering Events

Step 4: Plotting

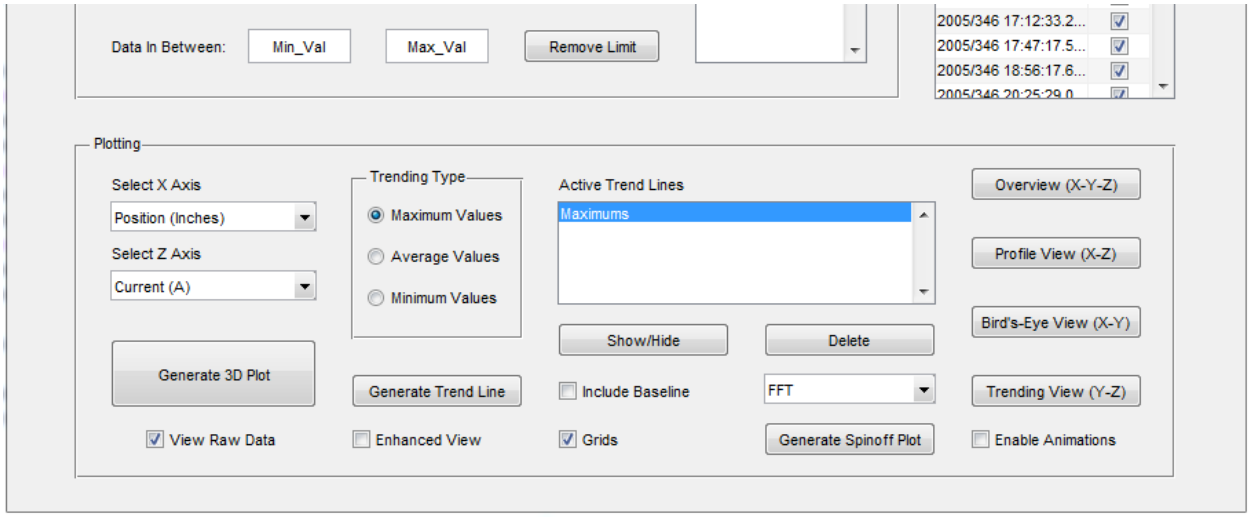


Figure 8: Plotting

The first step in visualizing the selected data is to create one or more 3D plots. To create a 3D plot, select the desired parameters for the X- and Z-axes. The Y-axis will automatically be populated with the dates of the selected events.

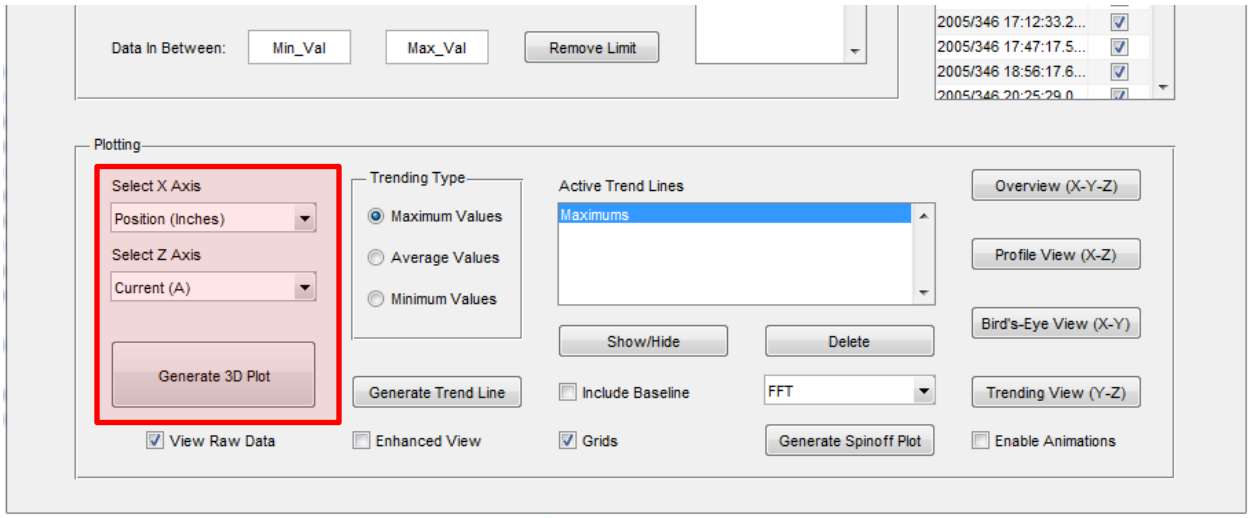


Figure 9: Axis Selection, Plot Generation

Once a 3D plot has been generated, there are multiple enhancement features available. The user may also update the filters at any point, which will automatically update the open figures. Note that the plots can always be modified from the command line or by using the standard MATLAB figure tools. However be careful in manually editing plots, as many features such as the title, legend, and axes limits are reset when certain GUI buttons are hit. For a complete description of figure updating functionality, see Appendix C: Figure Updating Functionality.

Feature A: Changing Views

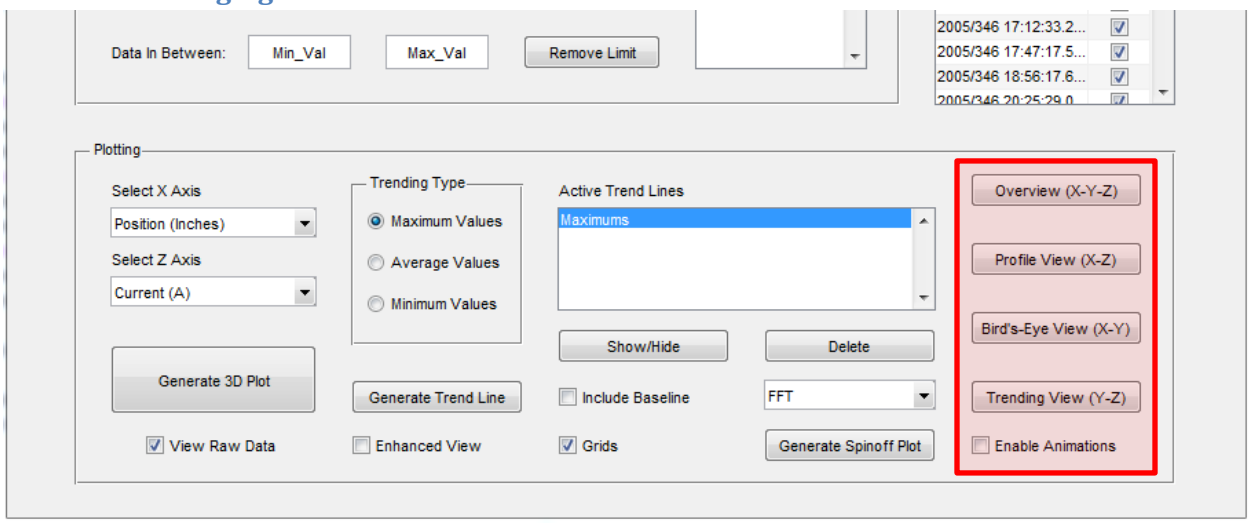


Figure 10: Changing Views

Although it is always possible to change the view with the rotation tool in the MATLAB figure window, these buttons provide a quick snap-to-view feature. With animations enabled, the figures will automatically rotate to the desired view, providing the user a better intuitive/contextual sense of the raw data. The views are described as follows with [Azimuth, Elevation] angle pairs:

1) **Overview**: [-25, 40]. This provides a summary view, looking at the data from an angle.
2) **Profile View**: [0, 0]. This shows a view of the X-Z-axes plane. It shows the shape of the data, but requires further enhancements to see how it is changing over time.
3) **Bird's-Eye View**:
    a. [0, 90] when **Enhanced View** is deselected. This provides a view of the X-Y-axes plane and is useful for visualizing the range of the x axis parameter over time.
    b. [90, -90] when **Enhanced View** is selected. This provides a similar view, but is more suited to viewing change in the X-axis parameter of active trend lines over time.
4) **Trending View**: [90, 0]. This view of the Y-Z-axes plane is suited for viewing changes in the z parameter range over time, or the z axis parameter of active trend lines over time.

View	[Azimuth, Elevation]
Overview	[-25, 40]
Profile	[0, 0]
Bird's Eye	Enhanced [90, -90], Not Enhanced [0, 90]
Trending	[90, 0]

Table 1: View Options

Feature B: Trend Lines

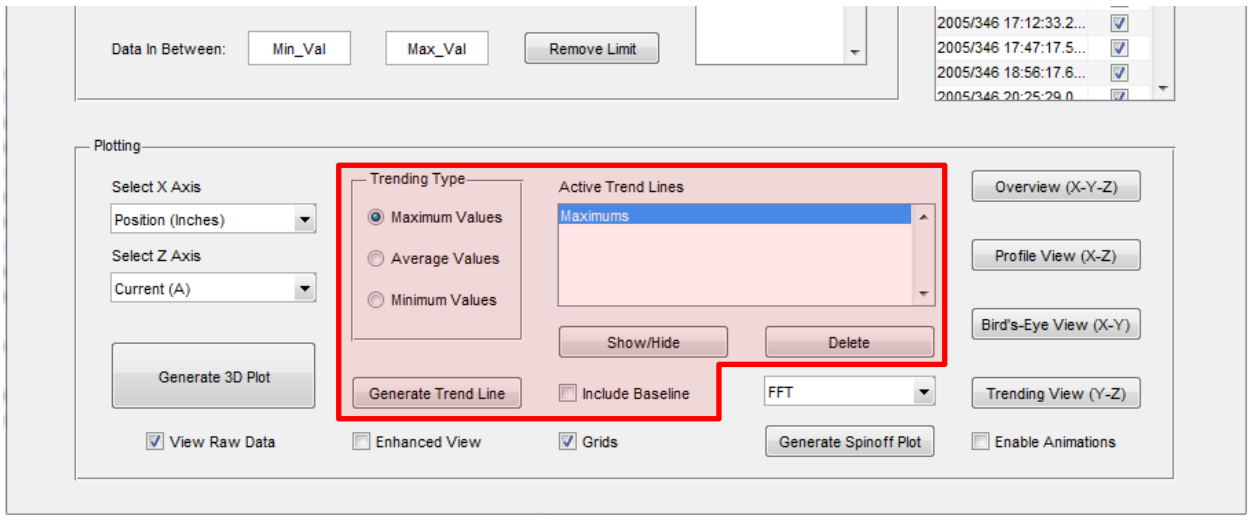


Figure 11: Trend Lines

Trend lines automatically display the maximum/average/minimum values on all open 3D plots. The maximum and minimum options will show the maximum of the z-parameter and the x-parameter at that point. Combined with the **Bird's-Eye View**, this can be used to view data such as position at peak current. The **Average Trend Line** will display the average z-parameter and the average x-parameter. Once a trend line is created, its values will not change, even if the filters are updated. This allows the user to view comparisons with different filters being active (Ex. peak currents of LEE A vs. LEE B).

The three buttons on the right provide the ability to temporarily hide or permanently delete trend lines. The **Include Baseline** checkbox will toggle the visibility of the average and 3σ upper and lower bounds for all visible trend lines.

Feature C: Additional Plot Settings

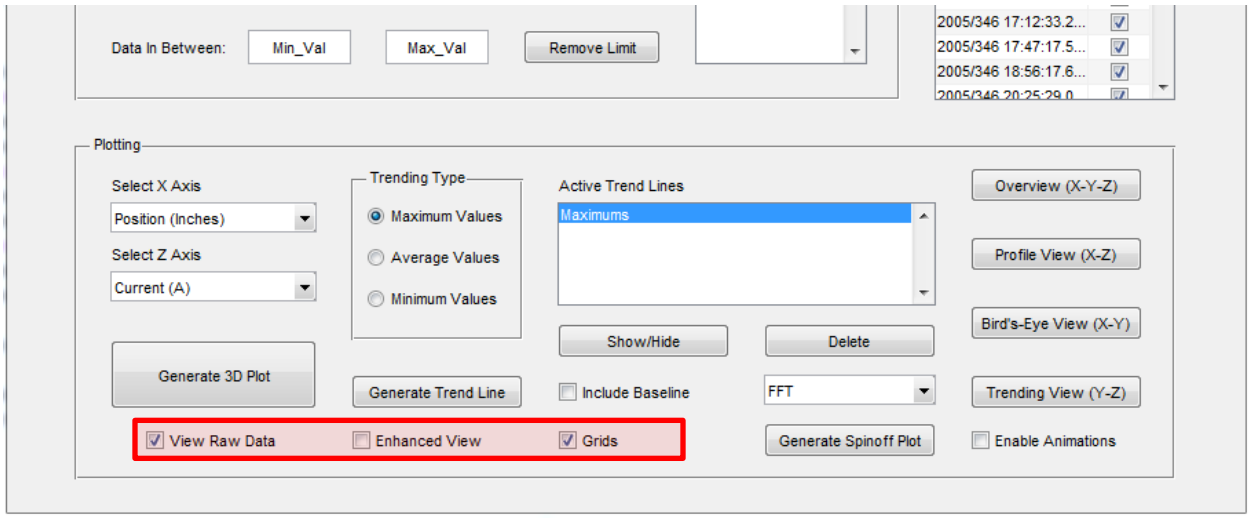


Figure 12: Additional Plot Settings

These three highlighted checkboxes allow the user to modify certain plot settings to improve its quality for presentations/documents. The raw data visibility can be toggled on/off if it is distracting from visible trend lines. The x- and z-grids can be toggled with the **Grids** checkbox. The **Enhanced View** checkbox will rotate the dates on the Y-axis, readjust the size of the plot within the window, and add year separation lines and labels that act as a Y-axis grid.

Feature D: Spinoff Plots

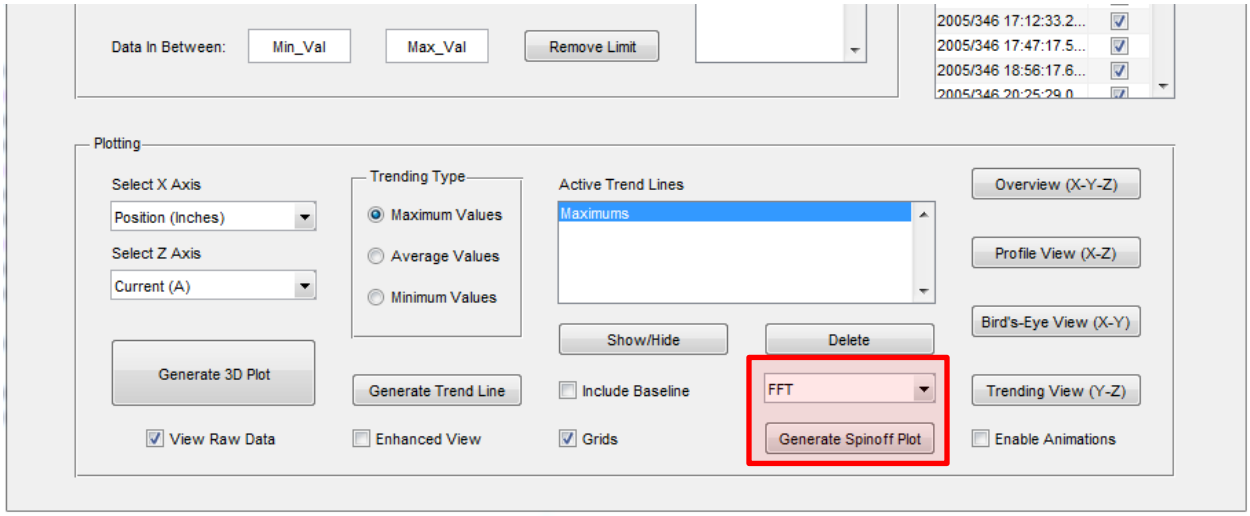


Figure 13: Spinoff Plots

Over the course of previous trending efforts, several types of analysis have proved to be useful. The **Generate Spinoff Plot** feature provides the ability to generate additional plots with information that are not for being displayed on the same axes as the raw 3D data. The selected data is passed to external functions of the MATLAB file format "*Spinoff_Plot_... .m*".

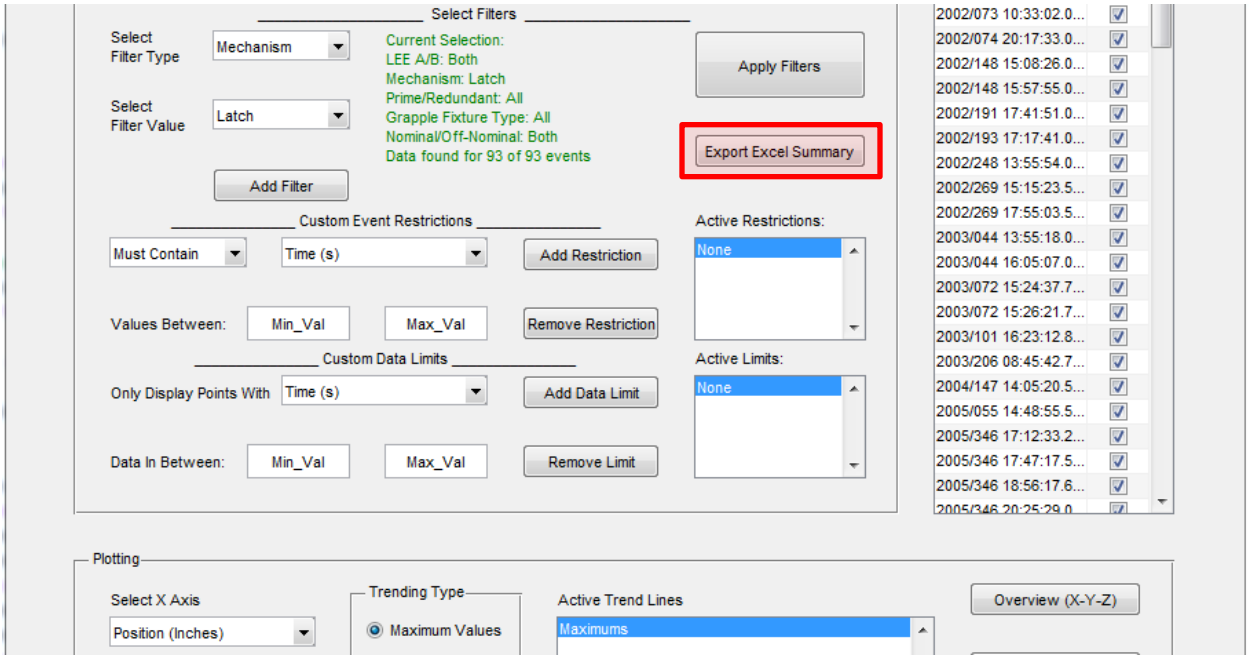
The **Spinoff** plots currently implemented are:

1) **FFT**: This function performs a Fast Fourier Transform on the selected motor rate data.
2) **Standard Deviation Trends**: This plots the trend of the standard deviations of the selected and plotted data. It will include baselines in the spinoff plot if the Include Baselines checkbox is selected.
3) **Trend of the Slopes**: This function will plot the slope of the line of best fit for the selected and plotted data over time.
4) **Difference of 2 Trend Lines**: This function will plot the difference of 2 trend lines over time. It is useful if it is suspected that one region is changing at a different rate than another region (Ex. average currents in second half of travel minus average currents in first half of travel). This function requires that at least 2 trend lines with identical dates have already been created.
5) **Temperature vs. Event**: This function plots the average temperatures vs. the selected events over time.
6) **Trend vs. Temperature Scatter Plot**: This function plots the values of the selected trend line vs. the average temperature for that event. It also includes lines of best fit, as specified by the user.

7) **Temperature Compensated Trend Line**: This function plots the selected trend line and a temperature compensated version of that trend line in a new figure. The compensation must be input manually each time, and the value may be extracted using the **Trend vs. Temperature Scatter Plot**.
8) **Smoothing Function**: This function approximates data sets, attempting to capture important patterns while leaving noise or fine-scale structures intact. The degree/order of filter used to smooth the function will be manually input as well as the cutoff frequency. This spinoff will generate a new chart exhibiting the X and Z features.

For more detailed information on these spinoff plots, see Appendix D: Spinoff Function Details. For information on creating new spinoff plots, see Appendix E: Creating a New Spinoff Function.

Feature E: Exporting an Excel Summary



This feature will create a Microsoft Excel file that contains all of the data points for every visible trend line. It will also include all custom filters, restrictions, and data limits information in the file.

3. Menu Bar Functionality

The menu bar provides additional load/save functionality and figure settings features.

The **Save Session** feature under the **File** menu allows the user to save the filter and trend line information from the current session as a .mat file. This .mat file can be loaded using the **Load Session** feature. The **3D Figure Settings** menu allows the user to change the style of the data lines being displayed. The options allow for multicolour lines with a Z-axis gradient, solid lines, or multicolour lines with a Y-axis gradient. The solid lines may also be modified to show only the points in a scatter plot. Several of these menu items have shortcuts. These shortcuts and additional hidden commands are described in Table 2: Key Commands.

Key Combination	Function
Alt+c	Cycles through the **C**olour scheme options. Default setting is 3D lines with a colour gradient that changes with the z axis value. Secondary setting is solid lines with a legend identifying the event. Tertiary setting is has solid lines with a gradient in the y axis, providing trending information.
Ctrl+s	**S**ave the current GUI session. This includes the processed data, all custom filters, restrictions, limits, manual event visibilities, and trend lines. It does not include any figure information.
Ctrl+l	**L**oad a previously saved GUI session.
Ctrl+d	**D**ebug. Copies all the processed data, GUI data, and figure data stored by the GUI to the MATLAB workspace. The GUI does not support reintegrating these variables, so any changes made in the workspace will not be reflected in the data stored by the GUI. For more information on the GUI data, see Appendix F: GUI Data Structures.

Table 2: Key Commands

Appendix A: Data Processing Function Details

The data processing scripts are called in step 2. They perform many functions such as eliminating erroneous data, creating filter information for the GUI, and specifying units. A detailed summary of each processing function is described in this section.

A.1 *process_SSRMS_LEE_data.m*

This function processes data from ops involving the SSRMS LEE snare, carriage, latch and umbilical mechanisms. The PUIs required in the text files are listed in the documentation at the beginning of this file and are repeated below (note that only 1Hz data was available before 2002/250):

A.1.1 Required PUIs in the Text Files

From GMT 2008/050 to present:

```
Active_Effector_Active_Mechanism    C_ActEffActiveMech
Active_Effector_Mechanism_Position  C_ActEffMechPos
Active_Effector_Measured_Motor_Current    C_ActEffMeasMotorCur
Active_Effector_Derived_Motor_Rate  C_ActEffDeriveMotRat
Computed_Active_Eff_Rigidization_Force_Derived_Torque C_ActEffRigFrcTorq
Operating_Base_OCS_SSRMS      CMRC13SW018DU
Capture_with_Latch_SSRMS      CMRC13SW00P0U
Fst_State_Ssrms_Primary_OCS   CMRC13SW018ZU
Fst_State_Ssrms_Redundant_OCS CMRC13SW0190U
Computed_Command_Status_ID_10Hz     C_CmdStatusID
```

From GMT 2002/250 to GMT 2008/050:

```
Active_Effector_Active_Mechanism    C_ActEffActiveMech
Active_Effector_Mechanism_Position  C_ActEffMechPos
Active_Effector_Measured_Motor_Current    C_ActEffMeasMotorCur
Active_Effector_Derived_Motor_Rate  C_ActEffDeriveMotRat
Active_Effector_Rigid_Force   C_ActEffRigidForce
Operating_Base_OCS_SSRMS      CMRC13SW018DU
Capture_with_Latch_SSRMS      CMRC13SW00P0U
Fst_State_Ssrms_Primary_OCS   CMRC13SW018ZU
Fst_State_Ssrms_Redundant_OCS CMRC13SW0190U
Computed_Command_Status_ID_10Hz     C_CmdStatusID
```

From start of life (2001) to GMT 2002/250:

```
Measured_Motor_Current_SSRMS_LEE    CMRC13SW00P4C
Derived_Motor_Rate_SSRMS_LEE  CMRC13SW00P3R
SSRMS_LEE_Rigidization_Force  CMRC13SW00OUG
Latch_Position_SSRMS_LEE      CMRC13SW00P7H
Carriage_Position_SSRMS_LEE   CMRC13SW00P6H
Snare_Position_SSRMS_LEE      CMRC13SW00P8H
Snare_in_Motion_SSRMS_LEE     CMRC13SW00OHU
Carriage_in_Motion_SSRMS_LEE  CMRC13SW00OKU
Latches_in_Motion_SSRMS_LEE   CMRC13SW00OPU
Umbilical_In_Motion_SSRMS_LEE CMRC13SW00ORU
Operating_Base_OCS_SSRMS      CMRC13SW018DU
Capture_with_Latch_SSRMS      CMRC13SW00P0U
Fst_State_Ssrms_Primary_OCS   CMRC13SW018ZU
```

```
Fst_State_Ssrms_Redundant_OCS CMRC13SW0190U
Computed_Command_Status_ID_10Hz     C_CmdStatusID
```

The optional PUIs that are also useful are:

```
Shell_Temperature_SSRMS CMRC13SW00U3T
Latches_Temperature_SSRMS     CMRC13SW00TZT
Carriage_Temperature_SSRMS    CMRC13SW00TYT
SMM_Motor_Temperature_SSRMS   CMRC13SW00U2T
RMM_Motor_Temperature_SSRMS   CMRC13SW00U1T
LMM_Motor_Temperature_SSRMS   CMRC13SW00U0T
Calibration_Status_Ssrms_Tip_Lee_OCS      CMRC13SW01BYU
Execution_Run_Speed_SSRMS     CMRC13SW00OYU
Ssrms_Gf_Id_SSRMS_Evnt  CMRC13SW01VTU
Latches_Disengaged_SSRMS_LEE  CMRC13SW00OMU
SSRMS_Base_Location_OCS_SSRMS CMRC13SW018CU
```

A.1.2 Raw Data Corrections

The optional PUIs allow for temperature analysis and are useful for the Excel summary. Events that do not contain temperature information will be omitted from temperature analysis. For each of the temperature PUIs found, the average of all values (not including 999999999 data) is taken as the temperature value for that PUI for that event.

Several data corrections are performed to get consistent results. The following corrections are applied to data from events prior to 2002/250:

1) Rigidization force data is converted from Newtons to lb·f.
2) The active mechanism is determined based on _in_Motion_SSRMS PUIs.
3) 1Hz position data from the different mechanisms are merged into a single data column using the active mechanism information.

The following corrections are applied to data from events occurring between 2002/250 to 2008/050:

1) Force data from "*C_ActEffRigidForce*" is converted to lb·f using the formula:

$$\text{Force (lb·f)} = C\_ActEffRigidForce \text{ data} \cdot 9.763653125 + 752.2539$$

The following corrections are applied to data from events occurring between 2008/050 and 2014/250

1) Current data is multiplied by a factor of 16.

No corrections applied to data post 2014/250.

In addition to time specific corrections, all data must satisfy the following conditions:

1) Position data must all be non-zero.
2) Current, motor rate, rigidization force, and position data must all be less than 100000 (filters out 999999999 type data).

3) An *Active_Effector_Active_Mechanism* value of 0 that occurs fewer than ten times in a row is typically faulty data. If the values before and after this are the same, the values of 0 are replaced. Otherwise those data points are removed.

Events are also sorted by time before being passed back to the GUI.

A.1.3 Filters Implemented

The following filters and related information are implemented by this script:

1) Lee A/B: The value of *Operating_Base_OCS_SSRMS* (CMRC13SW018DU) is checked.
    a) If it has a value of 0, LEE B is the tip.
    b) If it has a value of 5, LEE A is the tip.
    c) If both values 0 and 5 are found, the most frequent occurrence determines the LEE tip
    d) If neither are found, the tip LEE is unknown.
2) Mechanism: From 2002/250 onward *C_ActEffActiveMech* is checked. (…*_in_Motion_SSRMS_LEE* PUIs are used prior to this date).
    a) A value of 0 corresponds to the snare mechanism.
    b) A value of 1 corresponds to the carriage mechanism.
    c) A value of 2 corresponds to the latch and umbilical mechanisms (note that they are not distinguishable from the perspective of the processing script. Only the title text changes between the two).
3) Prime/Redundant: *Fst_State_Ssrms_Primary_OCS* (CMRC13SW018ZU) and *Fst_State_Ssrms_Redundant_OCS* (CMRC13SW0190U) are checked.
    a) If only one has a value of 5, that string is active.
    b) If neither have a value of 5, the operational string is unknown.
    c) If both have a value of 5, an error is returned and processing fails.
4) Grapple Fixture Type: The values of Capture_with_Latch_SSRMS (CMRC13SW00P0U) are checked.
    a) If a value of 1 appears more frequently than a value of 0, it will be filtered as a PDGF capture.
    b) Otherwise it will be filtered as a FRGF capture.
    c) If neither 0 or 1 is found, the GF type will be considered unknown.
5) Nominal/Off-Nominal: *C_CmdStatusID* and the Rigidization Force PUIs are checked.
    a) The last value of *C_ CmdStatusID* that is equal to 4, 7, or 11 is found.
        i. If the last value is 4 (rejected) or 7 (aborted), the file is filtered as Off-Nominal.
        ii. If none of those 3 values is found, an error is returned and processing fails.
    b) If the rigidization force drops by 100lb·f or more after reaching peak force, the file is filtered as Off-Nominal.
    c) If the peak rigidization force is not between 1210lb·f and 1272lb·f, the file is filtered as Off-Nominal.
    d) If there is a data gap that lasts longer than 3 seconds, the file is filtered as Off-Nominal
    e) If the file passes checks a) through d), it is filtered as Nominal.

A.2 *process_OTCM_data.m*

This function processes data from ops involving the SPDM OTCM gripper, advancer, torquer and umbilical mechanisms. The PUIs required in the text files are listed in the documentation at the beginning of this file and are repeated below:

A.2.1 Required PUIs in the Text Files

```
The expected (mandatory) PUIs in raw_data are as follows:

Active_Effector_Active_Mechanism    C_ActEffActiveMech
Active_Effector_Mechanism_Position  C_ActEffMechPos
Active_Effector_Measured_Motor_Current    C_ActEffMeasMotorCur
Active_Effector_Derived_Motor_Rate  C_ActEffDeriveMotRat
Computed_Meq_ID_(10Hz)  C_MeqID


The optional PUIs that are also useful are:

SPDM_Arm_2_OTCM_Housing_Temp_2      CMRC13SW0QG5T
SPDM_Arm_2_OTCM_Housing_Temp_1      CMRC13SW0QG4T
SPDM_Arm_1_OTCM_Housing_Temp_2      CMRC13SW0Q41T
SPDM_Arm_1_OTCM_Housing_Temp_1      CMRC13SW0Q40T
SPDM_Arm_2_OTCM_Torquer_MM_Rate     CMRC13SW0QC8R
SPDM_Arm_2_OTCM_Umbilical_MM_Rate   CMRC13SW0QC6R
SPDM_Arm_2_OTCM_Advance_MM_Rate     CMRC13SW0QC4R
SPDM_Arm_2_OTCM_OEU_Gripper_MM_Rate CMRC13SW0QC2R
SPDM_Arm_1_OTCM_Torquer_MM_Rate     CMRC13SW0Q04R
SPDM_Arm_1_OTCM_Umbilical_MM_Rate   CMRC13SW0Q02R
SPDM_Arm_1_OTCM_Advance_MM_Rate     CMRC13SW0Q00R
SPDM_Arm_1_OTCM_OEU_Gripper_MM_Rate CMRC13SW0PZYR
```

A.2.2 Raw Data Corrections

The optional PUIs allow for temperature analysis and are useful for active OTCM determination. Events that do not contain temperature information will be omitted from temperature analysis. For each of the temperature PUIs found, the average of all values (not including 999999999 data) is taken as the temperature value for that PUI for that event.

To be considered valid data, each point must satisfy the following conditions:

1) Position data must all be non-zero.
2) Current, motor rate, and position data must all not be equal to 999999999.
3) An *Active_Effector_Active_Mechanism* value of 0 that occurs fewer than ten consecutive times is typically faulty data. If the values before and after this are the same, the values of 0 are replaced. Otherwise those data points are removed.

Events are also sorted by time before being passed back to the GUI.

A.2.3 Filters Implemented

The following filters and related information are implemented by this script:

1) **OTCM 1/2**: The value of *C_MeqID* is checked.
    a) If a value of 6 is found, and no value of 7 is found, OTCM 1 is active.
    b) If a value of 7 is found, and no value of 6 is found, OTCM 2 is active.
    c) If neither 6 nor 7 are found, the file is removed.
    d) If both 6 and 7 are found, the motor rates are checked for non-zero values until the active mechanism is determined. The script will try, in order: Gripper rates, Advancer rates, Torquer rates, Umbilical rates.
2) **Mechanism**: The content of *C_ActEffActiveMech* is checked.
    a) A value of 0 corresponds to the gripper mechanism.
    b) A value of 1 corresponds to the advancer mechanism.
    c) A value of 2 corresponds to the torquer mechanism.
    d) A value of 3 corresponds to the umbilical mechanism.
3) **Advancer Slopes**: This filter searches for loaded portions of advancer travel. Due to noisy data, this filter has the potential for further optimization. It is defined as follows:
    a) The end of this region is defined as occurring 0.1cm before the current rises 0.1A above the negative peak current.
    b) The start of this region is defined as 0.5cm after the current drops below -0.2A
    c) The length of this region must be at least 0.7cm long.
4) **Gripper Secondary Peaks**: This filter searches for the secondary peak on slow micro grips. This filter is defined as follows:
    a) Gripper data must contain a minimum current below -0.8A, position data greater than 1.5cm, and must not contain a motor rate data exceeding -200rad/s.
    b) The start of this region is defined as the first point after the negative peak that is increasing in magnitude.

Appendix B: Adding a New Data Processing Function

The data processing functions take in the raw data extracted by the GUI (from handles.raw_data) and return the processed data (in *handles.data*). The structure of the input *raw_data* is described in section F.1 *handles.raw_data*. The required structure of the processed data is described in detail in section F.3.1 Processing Function Generated Elements of *handles.data*.

The steps required to add another processing function to the GUI are as follows:

1) Use ctrl-f to search for the following line of code in *Trending_GUI.m*:

   ```
   set(handles.System_selection,'String',{...
   ```

   Append the name of the processing function to the list that follows this line. Keep track of the index of the appended item.

2) Search for the following line of code (~line 180):

   ```
   system = get(handles.System_selection,'value');
   ```

   Add a new case to the switch statement with the same format as the existing cases, using the new processing function.

3) Using the existing processing scripts as a guide, ensure that the inputs and output match the format expected by the GUI. The *handles.Processed_status* is the handle to the text object in the GUI, and the processing function may use this to display the status of the processing.

Appendix C: Figure Updating Functionality

This section provides a description of how each GUI button modifies the open plots. There are two classes of buttons: those that modify the figure settings and those that modify the figure data.

C.1 Figure Settings Updating

The buttons that modify the figure settings are:

1) Any of the checkboxes in the manual event filtering table.
2) The **View Raw Data** checkbox.
3) The **Enhance View** checkbox.
4) The **Grids** checkbox.
5) The **Include Baseline** checkbox.

When any of these buttons are toggled, all of the following are updated for all open plots:

1) The visibility of all data lines, based on the event filtering table, the active colour scheme, and the event filtering table selections
2) The trend line and baseline visibilities. Note that the objects listed in the **Active Trend Lines** box and their baselines always exist whether they are visible or not.
3) The year separator visibility, year text visibility, Y-axis label orientation and axes position within the figure, based on the value of the **Enhance View** checkbox.
4) The legend, with legend priorities given in the following order:
    a) Highest priority: If the secondary colour scheme is active (no gradient), the legend will display the events
    b) If baselines are being displayed, the legend will show the trend lines and their bounds
    c) If any trend lines are being displayed, the legend will identify them
5) The X- and Z-axes grids.

C.2 Figure Data Updating

The following items regenerate figure data:

1) The **Apply Filters** button
2) The **Generate 3D Plot** button
3) The **Generate Trend Line** button
4) The **Show/Hide** button
5) The **Delete** button
6) A change in the colour scheme

When any of these buttons are toggled, all of the following functions are performed for all open figures:

1) The dates displayed in the Y-axis are updated based on automatic filters and visible trend lines
2) The titles are updated based on automatic filter selections
3) The X- and Z-axis labels are updated based on parameters selected (and only changes if units change)
4) Previous raw data objects, enhancement objects, trend lines and baselines are deleted and re-generated.
5) The axis limits are reset. If any parameter limits exist for data in the X- or Z-axes, the axes limits will reflect those limits. Otherwise, the limits are automatically set by MATLAB.
6) The contents of the **Active Trend Lines** box are reset.
7) The figure settings updates described in C.1 Figure Settings Updating are also performed.

Note that sometimes the automatic axis limits will be different for different views. If the axis limits are not ideal, clicking the **Show/Hide** button will reset the automatic axis limits based on the current view.

Appendix D: Spinoff Function Details

When a spinoff plot is generated, the data is selected by the GUI and then passed to an external script for further processing and plotting. Detailed operations of both the GUI and the external script implementations are discussed in this section.

D.1 FFT (*Spinoff_Plot_FFT.m*)

This spinoff function plots the Power Spectrum Density (PSD) of the Fast Fourier Transform performed on the motor rate of the selected data. This function expects to find parameters named '*Time*', '*Position*' and '*Motor Rate*.' If any one of these is not found, this function will not work. The expected input parameters for the spinoff script are set by the GUI and passed on. The spinoff plot describes input data format in its documentation.

The spinoff script will extract only the 10Hz data using the time parameter to select the data points closest the repeating 10Hz value. The longest segment that contains 10Hz data will be used.

Once the number of valid 10Hz data points is known for each event, the data is truncated so that each event uses the same number of data points (for consistency in the Fourier analysis). The truncation point is determined as follows:

1) If the shortest event contains at least half the number of data points as the longest event, all of the events will be truncated to the length of the shortest event and every event will be used in the FFT analysis.
2) Otherwise, events that fall under the 20th percentile in terms of length are omitted (>80% are kept). The length of the shortest event of those remaining is used to truncate the remaining events. The percent of events kept for analysis can be modified by changing the *percent_included* variable.

The script will then generate a 3D plot of the Motor Rate vs. Position (with an offset to start at position = 0) for the segments selected.

The signal passed into the MATLAB **fft()** function is the 10Hz motor rate with the DC offset removed, and a Hamming window applied. The PSD's frequency is converted from Hz to cycles/motor rev using the formula:

$$\text{Frequency [cycles/motor rev]} = \text{Frequency [Hz]} \cdot 2\pi / \text{Average motor rate [rad/s]}$$

D.2 Standard Deviation Trends (*Spinoff_Plot_Standard_Deviation_Trend.m*)

This spinoff will pass the Z-axis data to the external script. It will also pass the value of the **Include Baseline** checkbox. The spinoff script will calculate the standard deviation for every event. If the **Include Baseline** checkbox is active, the script will also include baseline values in the plot.

Baselines are calculated using the 3σ bounds (standard deviation of the standard deviation). If there are any points that lie outside of the bounds, they are marked as outliers and the bounds are recalculated, omitting the outliers.

D.3 Trend of the Slopes (*Spinoff_Plot_Slopes_Trend.m*)

Similar to the standard deviation spinoff, this option will pass the selected data (both X and Z parameters in this case) to the external script.

The external script calculates a line of best fit for each event, and the slope of this line is plotted against the date. This function works best with data that has been filtered by the processing scripts specifically for this purpose. To date (March 2015), only the Advancer slopes have been implemented for this purpose, but more may be created.

D.4 Difference of 2 Trend Lines (*Spinoff_Plot_Difference_of_2_Trends.m*)

This spinoff requires that there are at least two visible trend lines with identical dates. It will ask the user to select the minuend and subtrahend trend lines, which it passes to the external script. The external scripts will perform the subtraction and plot the results with typical trending enhancements.

D.5 Temperature vs Event (*Spinoff_Plot_Input_Trend_Basic.m*)

This spinoff passes the temperature values and dates to the external script, which simply plots it with typical trending enhancements. The user is asked to select from a list of valid temperature sensors.

D.6 Trend vs Temperature Scatter Plot (*Spinoff_Plot_Temperature_by_year.m* and *Spinoff_Plot_Temperature_by_event.m*)

This spinoff requires the user to select a visible trend line, a valid temperature sensor, and the type of line of best fit to display (by year or by event). It will execute once for every plot open since they show different trend values in the Z-axis.

If a line of best fit by year is selected, the external script creates a scatter plot of the trend values vs. the average temperature for that event. For each year, it also creates a line of best fit whose equation is displayed in the legend.

If a line of best fit by event is selected, the user must enter the number events to be used per line of best fit. If the number of events is greater than 50% of the total number of events, a single line of best fit is created in the scatter plot. Otherwise, the events are grouped by the selected number starting from the most recent event. Any remaining events are added to the first group.

Both of these methods display the equations of the lines of best fit, where the slope of representative line of best fit may be used for the temperature compensated trends in the following section.

D.7 Temperature Compensated Trend line (*Spinoff_Plot_Temperature_Compensated_Manually.m*)

This spinoff requires the user to select a visible trend line, a valid temperature sensor, and specify a correction factor to apply to that trend line. It will execute once for every plot open since they show different trend values in the Z-axis.

The external script will use the correction factor to normalize the trend to 0°C. It will plot both the original trend and the temperature compensated trend on the same axes.

D.8 Smoothing Function

(Spinoff_Plot_Smoothing_Function.m)

Appendix E: Creating a New Spinoff Function

To create a new spinoff function, an external function needs to be specified and the new spinoff needs to be added to the GUI code. The steps are as follows:

Step 1: Use ctrl+f to search for the following line of code (~line 1880) in *Trending_GUI.m*:

```
set(handles.Spinoff_selection,'String',{...
```

Append the desired name of the spinoff as it should appear in the GUI drop down menu to the list below that line. Keep track of the index for the appended item.

Step 2: Specify the data to pass to the external function. This should be done in the function (~Line 890) in *Trending_GUI.m* called:

```
function Spinoff_button_Callback(hObject, eventdata, handles)
```

Create a new "case" with the index of the item you created. Use the pre-existing cases as examples as needed. It is critical to understand the data structures described in Appendix F: GUI Data Structures in order to be able to select the desired data.

Step 3: If applicable, pass this data off to an external function with the name format: *Spinoff_Plot_[insert name here].m*

The existing spinoff plot functions may be used as examples if needed.

Appendix F: GUI Data Structures

To modify existing scripts, create additional support functions, or to debug a session, it is critical to understand the data structure in use. This section provides details to how the data is stored in the GUI.

The GUI saves a single variable called *handles* that contains all of the information required for the GUI to run. This variable has fields containing the handles to every object in the GUI including the buttons, text boxes, and menu items. These handles can be queried using the ***get()*** function and can be modified using the ***set()*** function. Further details on these functions as well as GUI objects can be found in the MATLAB documentation.

In addition to this long list of GUI object handles, the *handles* variable also contains the following user generated variables:

- *handles.import_dir*: This variable stores the location of the previously imported data and is only used by the **Import Raw Data** button.
- *handles.raw_data*: This variable contains the raw data imported when the **Import Raw Data** button callback function is executed. It is passed directly to the external processing functions and is not used after that. If the user loads a previous session, this variable will be empty which is why processing is not enabled with loaded sessions.
- *handles.figdata*: This variable contains the information of all GUI controlled 3D figures.
- *handles.colour_settings*: This variable contains the information on which colour setting is active.
- *handles.data*: This variable contains all of the processed data and trend data. It is created externally by the processing functions and is passed back into the GUI where it is further modified when the user executes specific callbacks through button presses.

F.1 *handles.raw_data*

Each element of this structure contains the information from a single text file in the following format:

- *handles.raw_data(i).data*: This contains all the numeric information from the text file. It does not include the first column of the text files that should contain the timestamps for each data point
- *handles.raw_data(i).textdata*: This contains all of the header information and the timestamps of each data point. Note that the columns of “data” and “textdata” are offset by one due to the timestamp column included in “textdata”.
- *handles.raw_data(i).title*: This variable contains the name of the text file as a string.

Note that the data and textdata fields are direct outputs of the **importdata()** function whereas the title field has been added in the GUI code.

F.2 *handles.figdata*

Each element of this structure contains the information needed to manipulate the GUI-generated 3D plots. The information is stored in the following format:

- *handles.figdata.figID*: The handle to the figure.
- *handles.figdata.axesID*: The handle to the axes of the figure.
- *handles. figdata.params*: The indices of the parameters being plotted.
- *handles. figdata.h3dlines*: The handles to the multicolour lines (generated using **surf()**).
- *handles. figdata.h2dlines*: The handles to the solid lines (generated using **plot3()**).
- *handles. figdata.enhanced_h_lines*: The handles to the lines separating the years.
- handles. figdata.enhanced_h_years: The handles to the year label text objects.
- handles. figdata.enhanced_h_y_txt: The handles to the rotated y axis label text objects.
- *handles. figdata.htrends*: The handles to the trend lines.
- *handles. figdata.hbaselines*: The handles to the trend line excluding outliers, the average value lines, the upper 3σ bound lines, the lower 3-bound lines, and the outliers (It's an [n × 5] array, where n is the number of trend lines).
- *handles. figdata.titletxt*: The contents of the title.
- *handles. figdata.xaxistxt*: The contents of the X-axis label.
- *handles. figdata.zaxistxt*: The contents of the Z-axis label.
- *handles. figdata.baseline_vals*: A [n × 4] array containing the values of the upper 3σ bounds, the average values, the lower 3σ bounds, and the number of outliers.
- *handles. figdata.trend_txt*: A [1 × n] cell array containing the name of each trend line.

F.3 *handles.data*

This structure contains all of the processed data information, filter options and information, and trend line information. It can be subdivided into information created by the processing function and information created by the GUI.

F.3.1 Processing Function Generated Elements of *handles.data*

A set of indices will be used for the different elements in this section. They are defined as follows:

- "i" is the index of the event (Ex. the i^{th} event occurred on GMT 2011/123)
- "j" is the index of the filter (Ex. the j^{th} filter selects which LEE is active)
- "m" is the index of the filter options (Ex. the m^{th} option of the j^{th} filter is LEE B)
- "k" is the index of the parameter (Ex. the k^{th} parameter is motor rate)
- "n" is the index of the data point (Ex. at the n^{th} data point, the current is 2A)
- "t" is the index of the temperature sensor (Ex. the t^{th} sensor is the RMM motor temperature)

The elements created by the processing functions are not modified by the GUI. They are:

- *handles.data.process_fcn*: This is a string that will be the start of every title for figures generated by the GUI. (Ex. 'SSRMS LEE')
- *handles.data.filters*: This contains all of the custom filter information. It is a [1 × j] structure that is further broken down as follows:
    - *handles.data.filters.name*: Contains a string of the j^{th} filter's name
        - Ex. *handles.data.filters(j).name = 'LEE A/B';*
    - *handles.data.filters.options*: A [1 × m] cell that contains the options for the j^{th} filter, as displayed in the GUI's **Select Filter Value** drop down list.
        - Ex. *handles.data.filters(j).options{m} = 'LEE A';*
    - *handles.data.filters.values*: A [1 × m] cell where each element contains a vector of possible values in filter data that satisfy the selected filter.
        - Ex. For LEE A selected, *handles.data.filters(j).values{m} = [1];*
        - Ex. For both LEEs selected, *handles.data.filters(j).values{m} = [1,2];*
    - *handles.data.filters.titletxt*: a [1 × m] cell that contains strings to use for the title when each filter value is selected. The title is created by concatenating the *process_fcn* with all the *titletxt* strings, in the order of the filters. This means that the filters should be in the order that makes sense for the title, and each string should include appropriate spacing
        - Ex. *handles.data.filters(j).titletxt{m} = ' A and B';* note the space before 'A'
- *handles.data.filterdata*: This is a [1 × i] cell array where each element contains an [n × j] numeric array with values that are compared to those in *handles.data.filters.values*. Some columns of the numeric array may be uniform (Ex. active LEE, since it does not change for a single event), whereas others may change (Ex. the active mechanism changes from snare to carriage to latch in a capture). For the n^{th} data point to pass the filtering, each of the j values must be one of the possible values in *handles.data.filters(j).values{m}*.
    - Ex. if *handles.data.filterdata{i}[n,j]* is 1, then LEE A is active at the n^{th} data point.
- *handles.data.parameters*: This contains all of the information on the parameters extracted from the text files. It is a [1 × k] structure that is further broken down as follows:
    - *handles.data.parameters.string*: A string containing the name of the parameter
        - Ex. *handles.data.parameters(k).string = 'Motor Rate';*
    - *handles.data.parameters.unit*: If the parameter maintains the same unit, this contains a string of the unit. Otherwise, it contains a cell of the possible units.
        - Ex. for motor rate, *handles.data.parameters(k).unit = 'Rad/s';*
        - Ex. for position, *handles.data.parameters(k).unit = {'Degrees', 'Inches', 'Inches', 'Inches'};*
    - *handles.data.parameters.unidep*: This indicates whether or not the unit is dependent on a filter. If it is not, the value is zero. If it is, the value corresponds to the index of the filter on which it is dependent.
        - Ex. for motor rate, *handles.data.parameters(k).unitdep = 0;*

- Ex. for position, *handles.data.parameters(k).unitdep = j;* where j is the mechanism filter. The filter value then becomes the index to select which unit to use. i.e:
  - *dep = handles.data.parameters(k).unitdep;*
  - *active_val = handles.data.filters(dep).values{m};*

  Note that a single value is expected. Filters that modify units must have a single number for each option *in handles.data.filters.values* (as of March 2015, this is only relevant for the mechanism filter).

  - *Unit = handles.data.parameters(k).unit{active_val};*

- *handles.data.parameterdata*: This is a [1 i] cell array where each element contains an [n × k] numeric array. These numeric arrays contain all the parameter data for that event (Ex. the current data, motor rate data etc.).
  - Ex. to get the current (k^{th} parameter) at the n^{th} data point for the i^{th} event:
  - *Current_val = handles.data.parameterdata{i}(n,k);*
- *handles.data.eventdates*: This is a [1 × i] cell array with each cell containing the timestamp string of the i^{th} event.
  - Ex. *handles.data.eventdates{i} = '2001/113 14:08:15.298';*
- *handles.data.temperature*: This is a [1 × i] structure containing the temperature information for each event. It is further broken down as follows:
  - *handles.data.temperature.name*: This is a [1 × t] cell array containing the names of the temperature sensors for a specific event.
    - Ex. *handles.data.temperature(i).name{t} = 'RMM Motor Temperature';*
  - *handles.data.temperature.value*: This is a [1 × t] numeric array containing the average temperature for all of the temperature sensors for a specific event.
    - Ex. *handles.data.temperature(i).value(t) = 5; (Degrees celcius);*

F.3.2 GUI Generated Elements of handles.data

The same indices will be used as in the previous section, with the addition of the following indices:

- "r" is the index of the restrictions (Ex. the r^{th} restriction requires that there are no motor rate values above 100 rad/s).
- "q" is the index of the limits (Ex. the q^{th} limit requires the current to be between 1A and 2A).
- "s" is the index of the trends (Ex. the s^{th} trend line shows the maximum values with the given filters).

The GUI generates additional elements as described below:

- *handles.data.full_params*: A [1 × k] cell array containing the names and units of the filtered data, suitable for use in axis labels.
- *handles.data.active_filter_vals*: A [1 × j] numeric array containing the 'm' values that were used to create the currently active filtered data.

    - Ex. *handles.data.active_filter_vals(j) = m;*
- *handles.data.selected_filter_vals*: A [1 × j] numeric array containing the 'm' values that are currently selected but not yet filtered in the GUI. Same form as *active_filter_vals*.
- *handles.data.active_restrictions*: A [r × 4] cell array containing information about the restrictions used to create the currently active filtered data.
    - The first column is a Boolean value indicating if the filtered data must (1) or must not (0) contain a specific data point.
    - The second column is the index (k) of the parameter that the restriction is being applied to.
    - The third column is the minimum value that must/must not appear in the data (an empty array if the minimum is not specified)
    - The fourth column is the maximum value that must/must not appear in the data (an empty array if the maximum is not specified)
        - Ex. to filter out all data that contains a motor rate value above 100rad/s: *handles.data.active_restrictions(r) = {0,k,100,[]};*
- *handles.data.selected_restrictions*: A cell array of the same form as *handles.data.active_restrictions*, with information about restrictions that are selected, but not yet filtered in the GUI.
- *handles.data.active_data_limits*: A [q × 3] cell array containing information about the limits used to create the currently active filtered data.
    - The first column is the index (k) of the parameter that the limit is being applied to.
    - The second column is the minimum value of the data to be displayed (an empty array if the minimum is not specified)
    - The third column is the maximum value of the data to be displayed (an empty array if the maximum is not specified)
        - Ex. to filter data to only have data points where the current is below 2A: *handles.data. active_data_limits (q) = {k,[],2};*
- *handles.data.selected_data_limits*: A cell array of the same form as handles.data. *active_data_limits*, with information about limits that are selected, but not yet filtered in the GUI.
- *handles.data.trends*: A [1 × s] structure containing information about selected trend lines. Each element of trends has the following elements:
    - *handles.data.trends.is_visible*: A Boolean value indicating if the trend line is visible on the plots or not.
    - *handles.data.trends.type*: A string indicating the trend type. Options are 'Maximums', 'Averages', or 'Minimums.'
    - *handles.data.trends.dates*: A cell array containing the timestamps of each event included in the trend line. Same structure as *handles.data.eventdates.*
    - *handles.data.trends.temperatures*: A structure containing the temperature information for each event included in the trend line. Same structure as *handles.data.temperature.*

- *handles.data.trends.filters*: A vector containing the filter values at the time that the trend line was created. Same structure *as handles.data.active_data_limits*
- *handles.data.trends.restrictions*: A cell array containing the active restrictions at the time that the trend line was created. Same structure as *handles.data.active_restrictions*
- *handles.data.trends.limits*: A cell array containing the active limits at the time that the trend line was created. Same structure *as handles.data.active_limits*
- *handles.data.trends.vals* A [k × i × k] numeric array containing the trend values. The first index is the parameter in the z axis whose peak is taken, the second index is the event, and the third index is the parameter at the point that the peak is taken. For average trend lines, the first index does not change the values.
  - Ex. Assume the k_c parameter is current and the k_p parameter is position. The peak current for the i^{th} event is *handles.data.trends(s).vals(k_c,i,k_c).* The position at that peak current would be *handles.data.trends(s).vals(k_c,i,k_p).*

- *handles.data.filtered_data*: This variable has the same structure as *handles.data.parameterdata*, but only contains values that passed the custom filters, restrictions and data limits (it includes data that has been deselected from the event visibility table).
- *handles.data.filtered_dates*: This variable has the same structure as *handles.data.eventdates*, but only contains events that passed the custom filters, restrictions and data limits (it includes data that has been deselected from the event visibility table).
- *handles.data.filtered_temperatures*: This variable has the same structure as *handles.data.temperature*, but only contains values from events that passed the custom filters, restrictions and data limits (it includes data that has been deselected from the event visibility table).
- *handles.data.omitted_dates*: This variable has the same structure as *handles.data.eventdates*, but only contains events that did not pass the custom filters, restrictions and data limits

Appendix G: MATLAB File Structure

The GUI operates by executing callback functions in *Trending_GUI.m* when the GUI is manipulated. The default properties and which functions to call are specified in *Trending_GUI.fig* and may be modified using MATLAB's GUIDE tool. In addition to the callbacks in *Trending_GUI.m*, there are also a handful of external functions that are called within specific callback functions. This is illustrated in Figure 14: MATLAB File Structure.

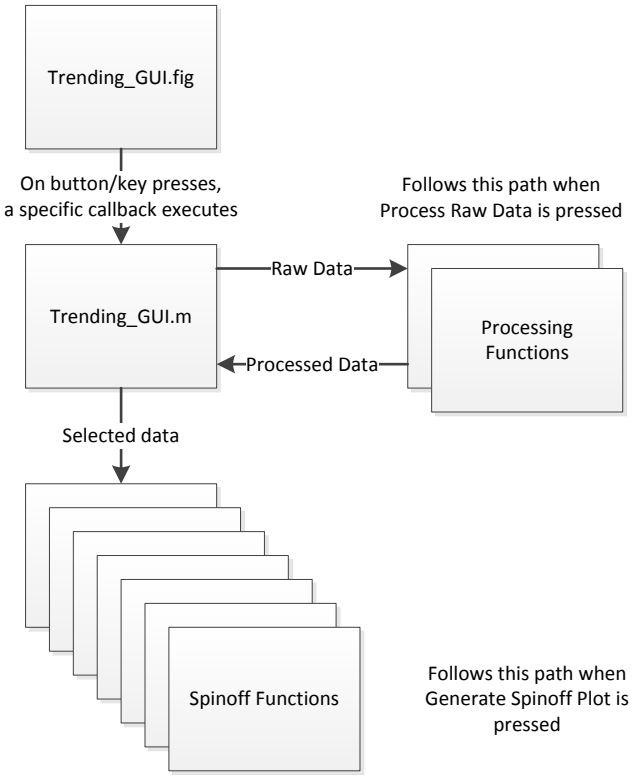


Figure 14: MATLAB File Structure

A table summarizing the callbacks implemented in *Trending_GUI.m* is shown in Table 3: *Trending_GUI.m* Callback functions summary.

Callback function name	Executes	Description
Raw_data_button_Callback	On **Import Raw Data** button press	Imports data from text files in the selected folders
Process_data_button_Callback	On **Process Raw Data** button press	Processes data using an external processing function
Filter_selection_Callback	On selection of an item from the **Select Filter Type** drop down menu	Modifies the options in the **Select Filter Value** drop down menu
Add_filter_button_Callback	On **Add Filter** button press	Modifies the selected custom filter values.
Apply_filters_button_Callback	On **Apply Filters** button press	Applies the selected custom filters, data restrictions, and

		data limits, in that order. Performs functions described in C.2 Figure Data Updating
Export_button_Callback	On **Export Excel Summary** button press	Exports an excel summary of the selected trend lines
Add_restriction_button_Callback	On **Add Restriction** button press	Adds the restriction to the selected restrictions list
Remove_restriction_button_Callback	On **Remove Restriction** button press	Removes the restriction from the selected restriction list
Add_data_limit_button_Callback	On **Add Data Limit** button press	Adds the data limit to the selected limits list
Remove_data_limit_button_Callback	On **Remove Limit** button press	Removes the data limit from the selected limits list
Event_table_Callback	On value change in the Event Visibility table	Performs functions described in C.1 Figure Settings Updating
Event_table_selection_Callback	On selection of elements in the Event Visibility table	Saves the selected elements to memory
Deselect_Group_context_Callback	On selection of "Deselect Group" in the right click context menu in the Event Visibility table	Disables the visibility of selected elements and performs functions described in C.1 Figure Settings Updating
Select_Group_context_Callback	On selection of "Select Group" in the right click context menu in the Event Visibility table	Enables the visibility of selected elements and performs functions described in C.1 Figure Settings Updating
Generate_3d_button_Callback	On **Generate 3D Plot** button press	Generates a new 3D plot and performs functions described in C.2 Figure Data Updating
Generate_trend_button_Callback	On **Generate Trend Line** button press	Creates a new trend line element and performs functions described in C.2 Figure Data Updating
Show_Hide_trend_button_Callback	On **Show/Hide** button press	Toggles the visibility of selected trend lines and performs functions described in C.2 Figure Data Updating
Clear_trend_button_Callback	On **Delete** button press	Deletes selected trend line element and performs functions described in C.2 Figure Data Updating
Baseline_check_Callback	On value change of the **Include Baseline** checkbox	Performs functions described in C.1 Figure Settings Updating
Raw_data_check_Callback	On value change of the **View Raw Data** checkbox	Performs functions described in C.1 Figure Settings Updating
Enhancedviewcheck_Callback	On value change of the **Enhanced View** checkbox	Performs functions described in C.1 Figure Settings Updating
Grid_check_Callback	On value change of the **Grids**	Performs functions described

	checkbox	in C.1 Figure Settings Updating
Spinoff_button_Callback	On **Generate Spinoff Plot** button press	Performs functions described in Appendix D: Spinoff Function Details
Overview_button_Callback	On **Overview (X-Y-Z)** button press	Changes views of all 3D plots as described in Feature A: Changing Views
Profile_button_Callback	On **Profile View (X-Z)** button press	Changes views of all 3D plots as described in Feature A: Changing Views
Trending_view_button_Callback	On **Bird's-Eye View (X-Y)** button press	Changes views of all 3D plots as described in Feature A: Changing Views
Birds_eye_button_Callback	On **Trending View (Y-Z)** button press	Changes views of all 3D plots as described in Feature A: Changing Views
Load_session_menu_Callback	On selection of **File → Load Session** from the menu	Performs functions as described in Menu Bar Functionality
Save_session_menu_Callback	On selection of **File → Save Session** from the menu	Performs functions as described in Menu Bar Functionality
Z_Gradient_menu_Callback	On selection of **3D Figure Settings → Colour Scheme → Z Gradient** from the menu	Performs functions as described in Menu Bar Functionality
Solid_Lines_menu_Callback	On selection of **3D Figure Settings → Colour Scheme → Solid Lines** from the menu	Performs functions as described in Menu Bar Functionality
Y_Gradient_menu_Callback	On selection of **3D Figure Settings → Colour Scheme → Y Gradient** from the menu	Performs functions as described in Menu Bar Functionality
Solid_line_line_menu_Callback	On selection of **3D Figure Settings → Solid Line Settings → Lines** from the menu	Performs functions as described in Menu Bar Functionality
Solid_line_scatter_menu_Callback	On selection of **3D Figure Settings → Solid Line Settings → Scatter Plot** from the menu	Performs functions as described in Menu Bar Functionality
Key_press	On any keyboard key press	Toggles through colour scheme with alt+c. Sends data to workspace with ctrl+d.

Table 3: Trending_GUI.m Callback functions summary