HIT3061 – Software Team Project - Semester 2, 2013

Tremor Detection with Leap Motion

Technical Manual

**Daniel Corsaletti**

SID: 6450458

E: 6450458@student.swin.edu.au

M: 0433 536 150

**Joshua Stopper**

SID: 5571391

E: 5571391@student.swin.edu.au

M: 0430 714 887

**Shengwei Li**

SID: 749999x

E: 749999x@student.swin.edu.au

M: 0420 478 750

**Minh Duc Nguyen**

SID: 171001x

E: 171001x@student.swin.edu.au

M: 0412 179 265

**Tran Xuong Tran**

SID: 6700691

E: 6700691@student.swin.edu.au

M: 0433 345 105

**Table 1. Document Change Control**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Changes |
| 1.0 | 15/10/13 | Ming | Created Document  Section WFLC |
| 1.1 | 6/11/13 | Joshua Stopper | Created Section 1 through 5 and completed content in each  Created table of contents  Structured document to conform to the same style as others |

Table of Contents

1.0 - Document Overview 4

2.0 - Application Overview 4

3.0 – System Requirements 4

3.1 – Leap Motion Controller Requirements 4

3.2 – Developed Application limitations 4

3.3 – Hardware and Software Recommendations 4

4.0 – Application Architecture 5

4.1 – File Structure 5

4.2 – File Purpose 6

4.3 – Application Data Flow 7

4.3.1 – File Load 7

4.3.2 – Introduction Modal 7

4.3.3 – Recording 7

4.3.4 – Analysis and Results 8

5.0 – Future Development 9

5.1 – Hand Visualiser 9

5.2 – Object Oriented or Prototypical code base 10

5.3 – Improved Data Store 10

5.4 – Noise Filtering 10

6 – Weighted Fourier Linear Combiner 12

# 1.0 - Document Overview

The purpose of this document is to introduce future developers to the “Tremor Detection with Leap Motion” application. The document includes a brief overview of the applications purpose and design, a discussion on the requirements and recommendations for use, an overview of the file structure and purpose as well as an explanation of how data flows through the application. Finally, recommendations are made for future development as well as outlining the potential benefits from the changes.

# 2.0 - Application Overview

“Tremor Detection with Leap Motion” is a single page application designed to interface with a Leap Motion Controller and analyse a set of captured frames to display attributes of tremor in the hand.

# 3.0 – System Requirements

## 3.1 – Leap Motion Controller Requirements

The Leap Motion website specifies the minimum system requirements for use of its controller. At time of writing the minimum system requirements are as follows:

* Windows® 7 or 8 or Mac® OS X 10.7
* AMD Phenom™ II or Intel® Core™ i3, i5, i7 processor
* 2 GB RAM
* USB 2.0 port
* Internet connection

*Note: For the most recent information, visit “http://support.leapmotion.com/entries/23822921-What-are-the-system-requirements-“*

## 3.2 – Developed Application limitations

Tremor Detection with Leap Motion requires the Google Chrome browser for the visual hand to be displayed properly. This is due to different stages of CSS implementation across different browsers. This is the only extra limitation imposed by the Tremor Detection with Leap Motion application.

## 3.3 – Hardware and Software Recommendations

When using the Tremors with Leap Motion application, it is recommended that the host device be directly connected to power. It was found that when the computer was not plugged in to power, the frame rate output from the Leap Motion Controller was drastically lower. This was evident when the application was capturing frames and the time taken to capture the required frames was extended.

It is also recommended that the latest version of Google Chrome be used as the browser on any operating system as the levels of CSS implementation vary and produce unexpected results when displaying a hand.

Finally, in the settings for the leap motion software running on the computer, we recommend that “automatic interaction height” be disabled and a manual height of 10.0cm is defined. By telling the drivers for the Leap Motion Controller what the interaction height is, several (internal to the driver) parameters can be tightened and optimised. The 10.0cm height was chosen as it was determined to be a good height to detect all fingers without losing track.

*Note: The height of 100mm is used as a base for determining in the "validFrame" function if the frame is valid. The variance in then applied in the positive and negative direction to this 100mm to give a valid range*

# 4.0 – Application Architecture

Tremor Detection with Leap Motion is a single page application, which uses existing libraries and API’s as well as custom programmed functions for frame recording and analysis.

## 4.1 – File Structure

The file structure for the application should be found as follows.

**/**

/Index.html

**/CSS/**

base.css

bootstrap.min.css

visualizer.css

**/js/**

tremors.js

**LeapMotion/**

colorPicker.js

extractData.js

handDisplay.js

lmController.js

validFrame.js

**AnalysisFunctions/**

getAccelerationAverage.js

getAmplitudeAverage.js

getArrayAverage.js

getEuclidean.js

getFrequencyAverage.js

getVelocityAverage.js

**libs/**

bootstrap.min.js

jquery.js

leap.min.js

*Note that folders have been bolded:*

## 4.2 – File Purpose

|  |  |  |
| --- | --- | --- |
| **File Name** | **File Location** | **File Purpose** |
| Index.html | / | Contains the base structure of the page. Contains the Modals, which are loaded at boot. Also loads all the required CSS and JS files |
| Base.css | /CSS/ | Contains the custom CSS developed for the page. Formats the panel of information on the right of the display |
| Bootstrap.min.css | /CSS/ | Bootstrap CSS file |
| Visualizer.css | /CSS/ | Contains the base formatting for the hand that is displayed on the screen |
| Tremors.js | /js/ | Contains basic event handlers for the DOM as well as captures frames from the leap motion device. |
| colorPicker.js | /js/LeapMotion/ | Used in conjunction with the Euclidean distance from the device, this changes the colour of the hand between red and green based on distance |
| extractData.js | /js/LeapMotion/ | Extracts data from the collected frames and formats them into to simpler arrays |
| handDisplay.js | /js/LeapMotion/ | Changes the position and orientation of the hand and updates the DOM. This is passed a leap motion frame |
| lmController.js | /js/LeapMotion/ | Interacts with the leap motion API and sends frames to tremors.js. |
| validFrame.js | /js/LeapMotion/ | Determines whether or not the current frame passed from the leap motion device is valid or not. Frames are valid depending on fingers and hands detected as well as position |
| getAccerlationAverage.js | /js/LeapMotion/analysisFunctions/ | Calculates the average acceleration of the data passed to it. This data comes from the return statement of “““““““““““““““““““““““““““““““““““““““““““““““““““““extractData””””””””””””””””””””””””””””””””””””””””””””””””””””” |
| getAmplitudeAverage.js | /js/LeapMotion/analysisFunctions/ | Calculates the average amplitude of the data passed to it. This data comes from the return statement of “““““““““““““““““““““““““““““““““““““““““““““““““““““extractData””””””””””””””””””””””””””””””””””””””””””””””””””””” |
| getArrayAverage.js | /js/LeapMotion/analysisFunctions/ | Calculates the array average of the data passed to it. This data comes from the return statement of "extractData" |
| getEuclidean.js | /js/LeapMotion/analysisFunctions/ | Calculates the Euclidean distance from a set point in virtual space. |
| getFrequencyAverage.js | /js/LeapMotion/analysisFunctions/ | Calculates the frequency of the data passed to it. This data comes from the return statement of "extractData" |
| getVelocityAverage.js | /js/LeapMotion/analysisFunctions/ | Calculates the average velocity of the data passed to it. This data comes from the return statement of “““““““““““““““““““““““““““““““““““““““““““““““““““““extractData””””””””””””””””””””””””””””””””””””””””””””””””””””” |
| Bootstrap.min.js | /js/libs/ | This is the JavaScript file downloaded as part of the bootstrap package |
| Jquery.js | /js/libs/ | This is the jQuery library |
| Leap.min.js | /js/libs/ | This is the JavaScript API provided by Leap Motion to interact with the Leap Motion Drivers installed on the computer |

## 4.3 – Application Data Flow

As described above, each of the files included has a specific purpose and is arranged in the folders are per their purpose. To have the files all operate together, each of the scripts and CSS files are included in the “index.html” document. The remainder of this section will attempt to give an idea of the flow of data through the application and its files.

### 4.3.1 – File Load

When the index.html file is opened, the required scripts and CSS files are loaded. Using the JQuery library and the “.ready()” method, the introduction modal is then opened. This can be seen in the “tremors.js” file at approximately line 18.

### 4.3.2 – Introduction Modal

The introduction modal introduces the user to the application and allows two operations, “begin” or “options”. The modals and their content can be viewed in the “index.html” file whilst the event handlers for all modals can be found in “tremors.js”.

The begin button will close the modal and start the recording. This can be seen in “tremors.js” at approximately line 25.

The options button will close the modal and open the options modal. The options modal contains several options that can modify the behaviour of the application. These options include but aren’t limited to recording time and fingers required in frame. These options are once again applied in the “tremors.js” file and are coded for display in the “index.html” file.

### 4.3.3 – Recording

Once begin recording has been clicked, a global Boolean variable named “recording” is set to true. Its default state is false. This informs the “frameController” function to record the passed in frame.

What is not mentioned earlier is that the Leap Motion API has been configured to send a frame to the application 60 times a second, this is configured in the “lmController.js” file. The “frameController” function checks if the application is set to record. If so, the current frame is sent to multiple functions.

The first function the frame is sent to is “displayHandsFingers”. This function resides in “handDisplay.js” and accepts a frame as a parameter to display the hands and fingers in the DOM.

The second function also accepts a frame as a parameter as well as the fingers required and updates the information panel on the right hand side of the display. This function is called “displayInfo”.

Thirdly, an “if” is executed to determine whether or not the frame contains valid data. The function that determines whether or not the frame is valid can be found in “validFrame.js”. A valid frame is defined as:

* Containing only 1 hand
* Containing the specified amount of fingers (configurable in the options menu)
* Being in the correct position (this is a range, this can be configured)
  + Vertical position
  + Horizontal positions
  + Depth position

If the frame is determined to be valid, a variable titled “preRecordFrames” is incremented by one. If the frame is determined to be invalid, the “preRecordFrames” and “recordedFrames” are reset to “0” and “[]” respectively.

The “preRecordFrames” variable is used to determine if 1 second (60 frames) of valid data has been recorded. It was found during testing that if recording of frames to the “recordedFrames” was started as soon as a valid frame was detected, the analysis results were less accurate due to the hand still having intended movement to get in the correct position. Thereby, informing the end user that the hand is in a valid position before recording actually begun, the user stopped their intended movements.

Next, the “recordedFrames” variable is tested to see if the required time (amount of frames) of valid frames has been collected. This is achieved by multiplying the time required by the amount of frames pushed from the device per second. If the required time has been reached, recording is stopped and the required data is extracted from the frames collected. This is handled by the “““““““““““““““““““““““““““““““““““““““““““““““““““““extractData””””””””””””””””””””””””””””””””””””””””””””””””””””” function.

### 4.3.4 – Analysis and Results

Once the “““““““““““““““““““““““““““““““““““““““““““““““““““““extractData””””””””””””””””””””””””””””””””””””””””””””””””””””” function has returned, the data is passed to the “updateResultsModal” function. The “updateResultsModal” performs two tasks. It uses the extracted data to calculate the hertz, amplitude, velocity and acceleration of each finger and then adds this information to a modal.

The “updateResultsModal” uses a “for loop” to iterate over each of the fingers stored in the passed in parameter. Inside this loop, the results from the following functions are added to an output variable that is then displayed:

* “getFrequency”
* “getAmplitude”
* “getVelocityAverage”
* “getAccelerationAverage”

Once the “for loop” has finished, the function ends and the parent displays the results modal. From there the user can view the results, configure options for the next recording or start a new recording.

# 5.0 – Future Development

Future development on “Tremor Detection with Leap Motion” can branch in several directions. The following section highlights the shortcomings in the development process so that improvements can be made.

## 5.1 – Hand Visualiser

Currently, the “Tremor Detection with Leap Motion” application works in the most current builds of the Google Chrome browser. Though extensive testing of versions was not conducted, it is recommended that the latest available build of Google Chrome be used.

The “Hand Visualiser” JavaScript and CSS files rely on 3D transformations being present and working in the browser. In browsers where these attributes are not accessible, the DOM will merely display several blocks in the middle of the screen. This is due to 3D transformations not being fully implemented across all browsers.

In the future, as browser support for 3D transformations improve, optimisation of the “visualizer.css” and “handDisplay.js” files can be made. This would include the addition/removal of browser specific prefixes as well as the potential for moving to a canvas based or 3.js implementation.

These changes should results in:

* Improved cross browser support
* Improved frame rate
* Greater control over appearance of objects

## 5.2 – Object Oriented or Prototypical code base

The current code base does not utilise entirely “best practice”, therefore a improvement could be made here. By moving to a object-oriented or prototypical code base, several noticeable improvements can be made, this could include:

* A reduction in the global footprint
* An improvement to readability of the code
* An improvement to extensibility of the code
* An improvement to modularity of the code

## 5.3 – Improved Data Store

The current code base currently uses a multidimensional array to store and describe extracted attributes of the hand. The current structure of the array is as “[finger][Attribute]” where each finger and attribute is an array that is indexed by an integer.

By using an object-oriented approach, the data store could be changed to a collection of objects (fingers) that hold all the attributes and methods available to that finger.

This would once again result in:

* A reduction in the global footprint
* An improvement to readability of the code
* An improvement to extensibility of the code
* An improvement to modularity of the code

## 5.4 – Noise Filtering

Noise filtering a tremor data set represents an interesting problem. Tremor is inherently up and down movement on the vertical scale. While recording this data is not a problem, analysing the attributes of tremor, most notably, amplitude requires that intentional movement be removed from the unintentional movement, thus noise filtering.

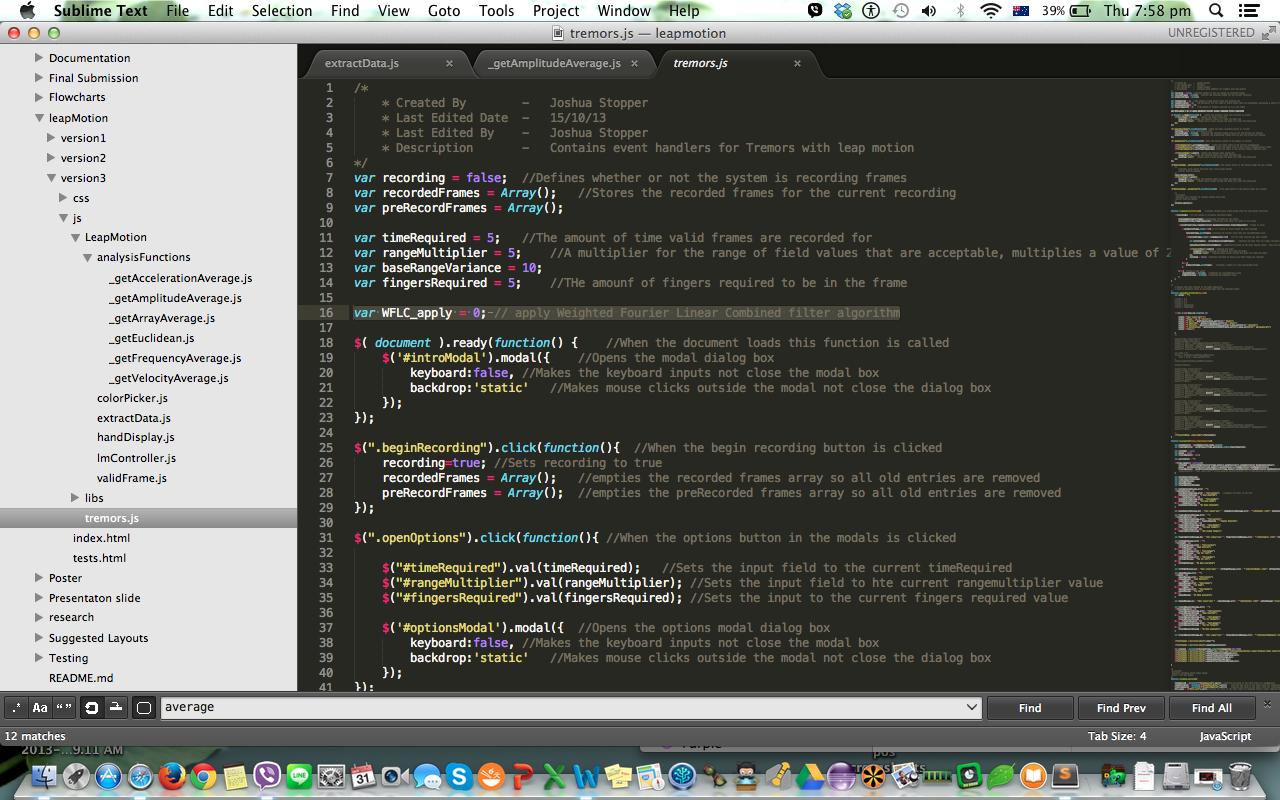
Noise filtering can be achieved in multiple ways. Several methods were discussed with the Philip Michael during the project. This included a “Weighed Fourier Linear Combiner” and “Fourier Linear Combiner”.

Ultimately neither method was implemented due to the complexity of the algorithms and the lack of documentation that fully explained their use. Noise filtering could improve the repeatability of the results returned by “Tremor Detection with Leap Motion”.

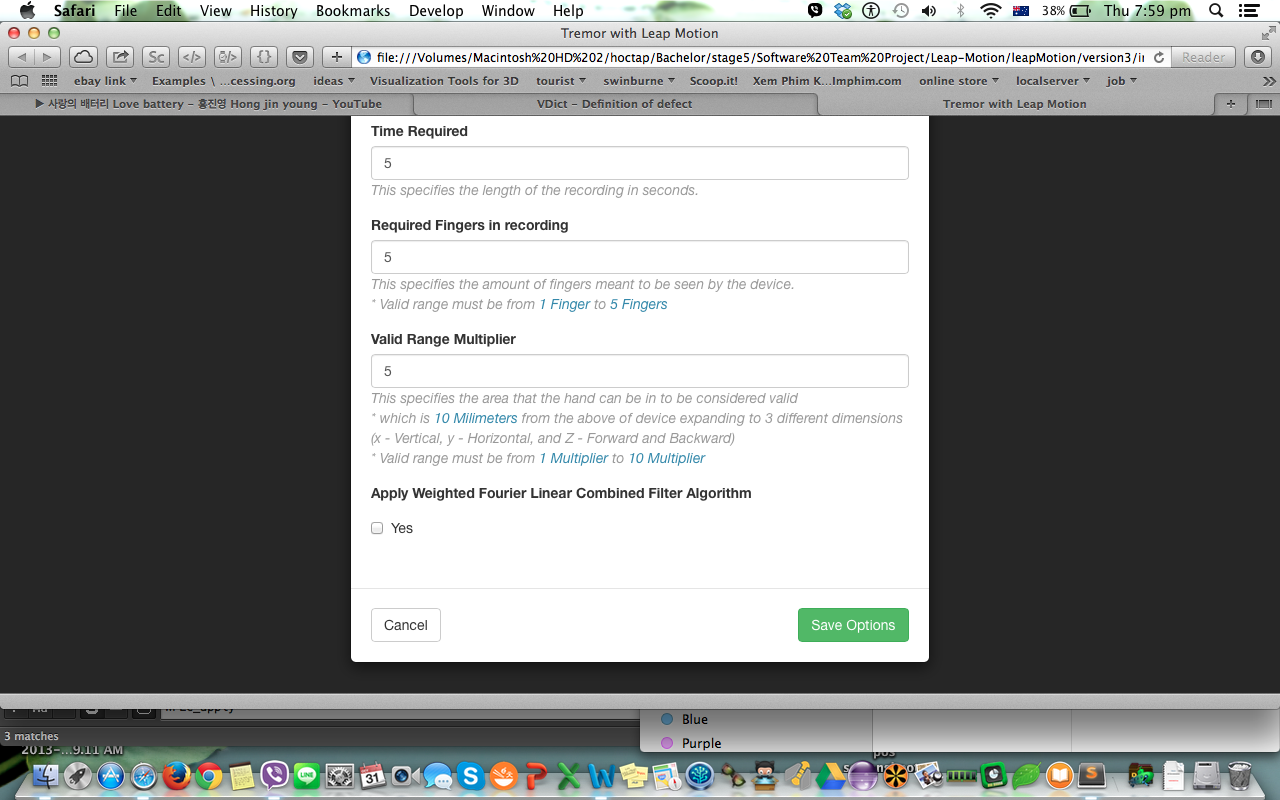
*Note: During data extraction from the frames, the Cartesian coordinates are limited to 1 decimal place. This limits the accuracy depth of the coordinate to 1 decimal place of a millimetre. For a proof on the accuracy of the leap motion device please see the document titled “Analysis of the Accuracy and Robustness of the Leap Motion Controller”*

# 6 – Weighted Fourier Linear Combiner

WFLC algorithm implementation flag variable “WFLC\_apply” is declared as global variable in “js/tremors.js” file line 16. The initial value of this variable is 0 indicating that WFLC algorithm is not applied.



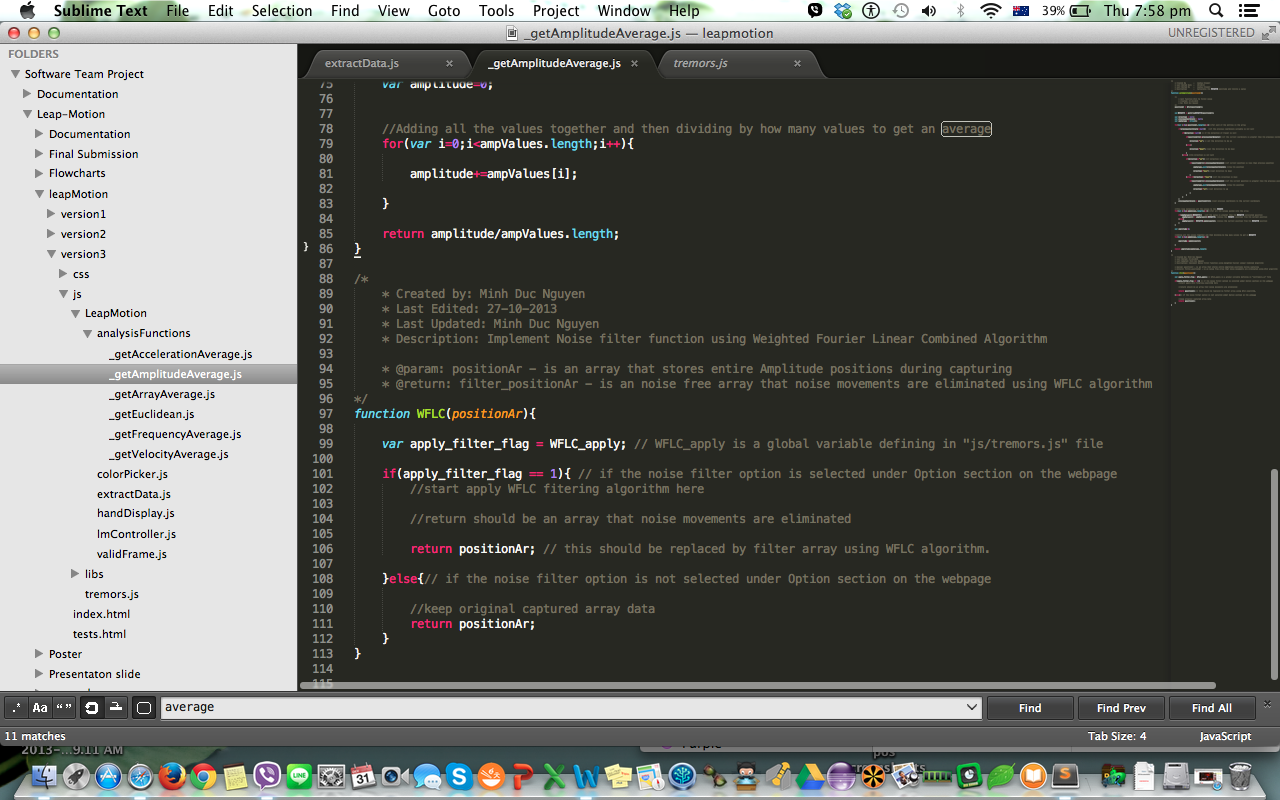
In the option modal in “index.php” page, the check box “Apply Weighted Fourier Linear Combined Filter Algorithm” is used to enable or disable WFLC algorithm in calculating output. The variable “WFLC\_apply” value is 0 by the default and will be set to 1 if the check box “Yes” is ticked.



In “js/tremors.js” file, line 385, this statement gets the current checkbox status (tick / un-tick) and assign to the global variable “WFLC\_apply”

## Macintosh HD:Users:ng0kylan:Desktop:Screen Shot 2013-10-31 at 7.59.18 pm.png

The WFLC implementation is defined in “js/LeapMotion/analysisFunctions/\_getAmplitudeAverage.js” file at line 97. The accepted parameter is an array object that stores a set of Amplitude values during the capture. The purpose of this function is to read the “WFLC\_apply” flag variable and apply WFLC filter algorithm if this flag value is 1 (selected on Option page as a checkbox). The output of this function is an array comprising a set of Amplitudes value that were cleaned using WFLC algorithm.



In “js/LeapMotion/analysisFunctions/\_getAmplitudeAverage.js” file line 14, this statement calls the “WFLC” function before process Average Amplitude output to the report on screen.

