[Algorithm to detect eye tracking events 2](#_Toc1556640)

[1. Inception 2](#_Toc1556641)

[2. Objective 2](#_Toc1556642)

[3. Introduction 2](#_Toc1556643)

[4. Apparatus 2](#_Toc1556644)

[5. Data Manipulation 3](#_Toc1556645)

[5.1. Data Preparation 3](#_Toc1556646)

[5.2. Data Filtering 4](#_Toc1556647)

[a. Raw data plot: 6](#_Toc1556648)

[b. Confidence Filtered 6](#_Toc1556649)

[c. Median Filter 6](#_Toc1556650)

[d. SGolay Filter 6](#_Toc1556651)

[e. SGolay then Median 6](#_Toc1556652)

[f. Median then SGolay 6](#_Toc1556653)

[6. Algorithm 8](#_Toc1556654)

[6.1. Blink Detection 8](#_Toc1556655)

[6.2. Fixation detection 12](#_Toc1556656)

[7. References: 14](#_Toc1556657)

# Algorithm to detect eye tracking events

## Inception

The development of this algorithm is a part of assignment for the course.

## Objective

The proposed algorithm will attempt to detect the following events:

1. Blink
2. Fixation
3. Saccades
4. Smooth Pursuit

First event being of higher priority.

This algorithm will make use of the raw data and available filters (see Data Filtering). I anticipate using the raw data and data post filter application to be able show hint of some events.

This algorithm will also attempt to make use of the DBSCAN algorithm, which clusters points according to density around the detected centroid. These clusters will be linked to their timestamps, to avoid clustering two mutually exclusive fixations as one, and try to validate the events detected previously.

## Introduction

Detecting events in eye tracking is a very interesting problem and this algorithm attempts to find them given the raw gaze data. Various filters are tested and applied. This algorithm also tries to use clustering algorithms.

## Apparatus

This experiment was conducted using the Pupil Eye Tracker. All calibrations and capturing were done in a semi-controlled environment. All data was processed and visualized on MATLAB. The first 25 seconds of recording was chosen to test the algorithm since it involved saccades, fixations and smooth pursuit in the following order.

Table 1 Relates which event occurred when in the first 25 seconds of the video for test data

|  |  |
| --- | --- |
| **Time frame (approx.)** | **Anticipated Event (from video observation)** |
| 0-3.3 seconds | Saccades and fixations |
| 3.0 seconds | Blink |
| 6.1 seconds | Blink |
| 6.2 – 25 seconds | Smooth Pursuit |

## Data Manipulation

### 5.1. Data Preparation

Before the data can be processed, it needs to be cleaned. For this, the confidence level of the data was used. Any data lower than a certain value must be omitted. The values chosen were 0.4 and 0.6. The comparison process is explained below. A plot with confidence of 0.9 was also added to observe extreme data.

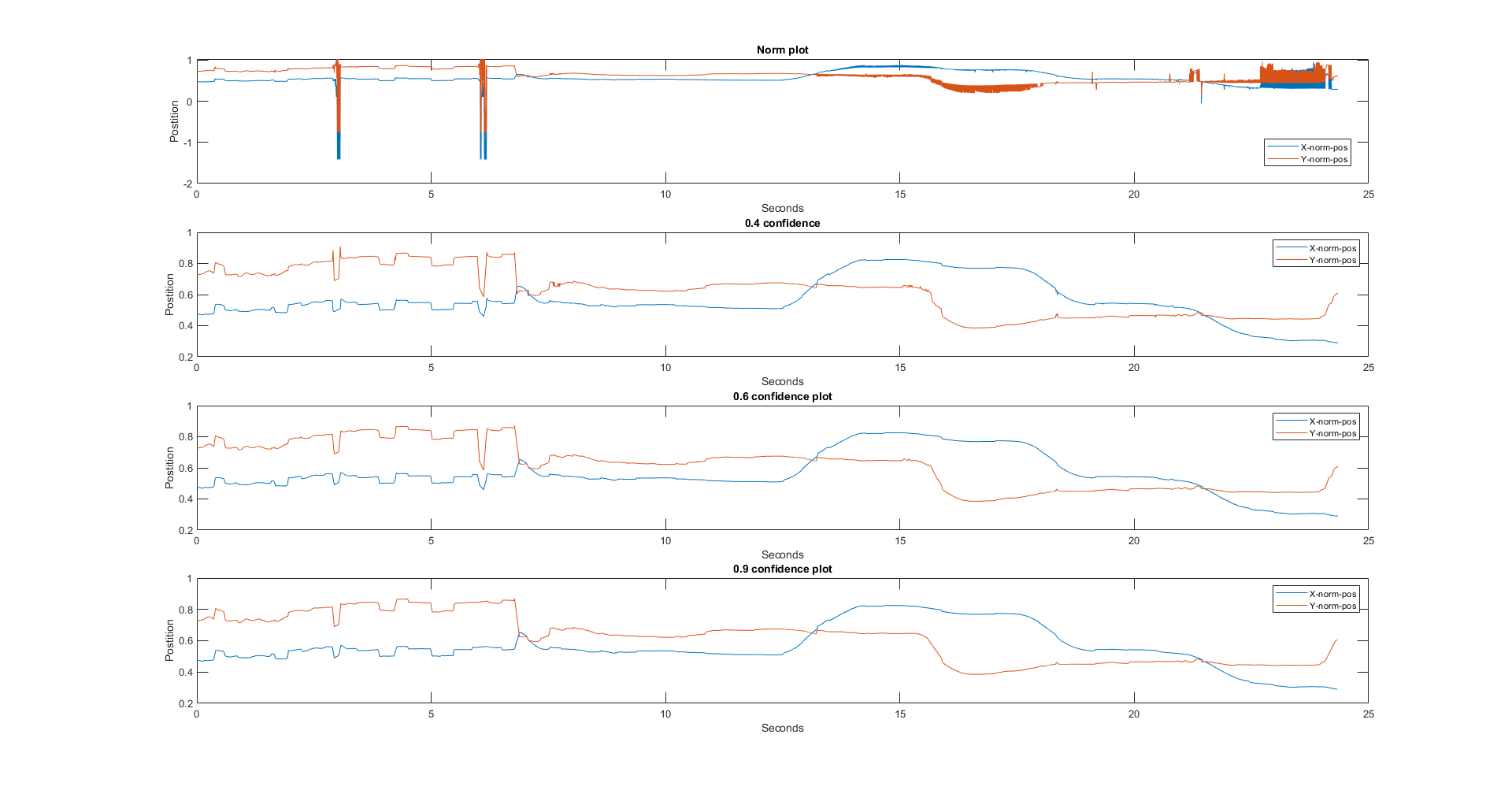


Figure 1 Comparison of confidence filters

In Figure 1 we observe three distinct differences. First (denoted by green oval), is that the blinks are smoothened out as we increase the confidence value. Hence, no confidence filtering is chosen for blink detection (Table 1). Second (black oval), the errors in smooth pursuit are smoothened out. This is preferable when detecting smooth pursuit. Finally, the noise from the smooth pursuit towards the end (pink oval) is also smoothened out. Therefore, filtering by confidence will allow us to have less noise when detecting smooth pursuit. Also note, that in the first case, the sections marked by double green ovals, we observe that the blink jitters have completely disappeared. Additionally, the orange rectangle shows that plots with confidence of 0.6 has less noise. From these observations, the confidence of 0.6 was chosen.

### 5.2. Data Filtering

The following algorithms chosen to filter data and reduce noise.

1. Savitsky-Golay Filter
   1. Power: 3
   2. Frame Size: 5
2. Median Filter
   1. Order: 3

Unless otherwise mentioned, the values above are used to filter data.

Initial comparison tests were conducted to choose one or more of these. The sections below discuss the observed effect of each filter, individually and combined.

In this trail Savitsky-Golay (hence referred as SGolay) filter and Median filter were chosen. There are three possible use cases for these filters. Only SGolay, only Median filter or both filters paired. All such possibilities are explored and discussed below. To determine effects of the filter, the parameters were pushed to extreme. The following values were used to test these filter effects:

1. Savitsky-Golay Filter
   1. Power: 7
   2. Frame Size: 13
2. Median Filter
   1. Order: 30

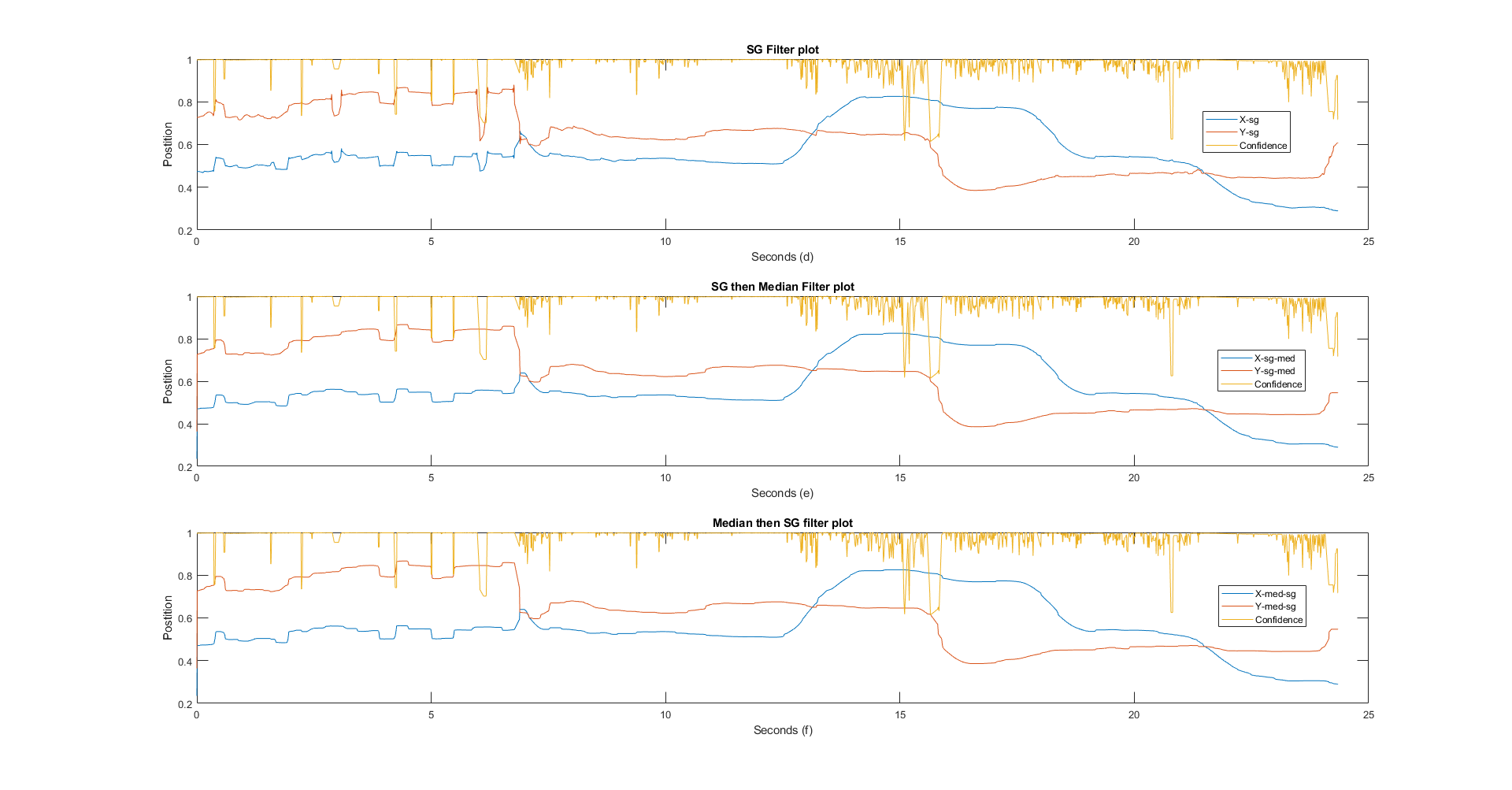
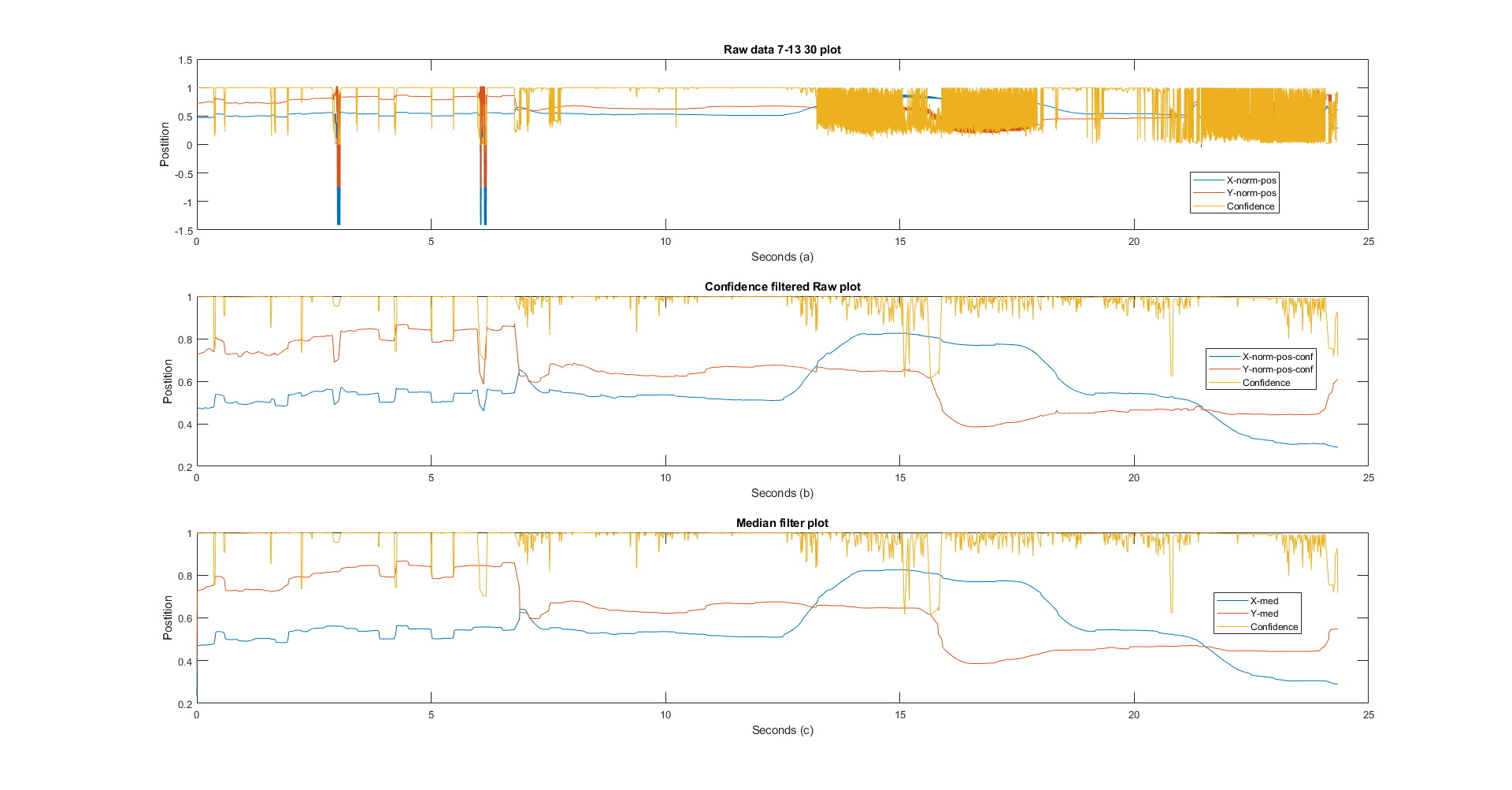
****

Figure 2 This figure shows the effect of different filters on the raw data. (a) plot of unfiltered raw data; (b) plot of data filtered by confidence of 0.6; (c) plot of raw data after it is passed through Median filter of order 3; (d) plot of raw data after it is passed through SGolay filter; (e) plot of SGolay filtered data after it is passed through Median filter; (f) plot of Median filtered data after it is passed through SGolay filter. One of the blinks is shown in green box and is referred in the discussion below.

##### Raw data plot:

Raw, unfiltered data is plotted along with confidence. After observation we see that the confidence drops every time the eye makes a saccade. Confidence also fluctuates when the data is noisy or when blinks (Table 1) are *detected.* The title informs us that the values here are as mentioned above.

##### Confidence Filtered

This plot is compared to the raw data plot (a). This plots the data after confidence filter of 0.6 has been applied. We can observe from this that the data on blinks (green rectangle) is presented as a trough, smaller and sharper than a saccade. Most of the data during the smooth pursuit (Table 1) phase is smoothened out. In all subsequent filters this data is used.

##### Median Filter

This plot is compared to the plot (b). In this we observe that the blinks are completely lost and presented as fixations. It does seem to retain the fixations and saccades well.

##### SGolay Filter

This plot is compared to the plot (b). From visual observation it is clear, that this retains the blinks as sharp troughs but at the same time alters it (green box). No other significant benefits of this filter over the Median filter are observed.

##### SGolay then Median

This plot is compared to the SGolay only plot (d). The Median filter again gets rid of the blinks. No other significant benefits of this filter over (d) are observed.

##### Median then SGolay

This plot is compared to the Median only plot (c). No other significant benefits of this filter over (c) are observed.

This ineffectiveness of SGolay filter is unusual. To check this, we pass unfiltered data to the SGolay filter with the parameters mentioned above. In Figure 3 we observe that SGolay and Median filters show different results when run on raw data and confidence filtered data (Figure 2 (c) & (d)). In Figure 3 we observe that SGolay filter retains the blinks and further smoothens the smooth pursuit part of the plot. Perhaps SGolay is a better choice for blink and smooth Pursuit detection. The Median filter relegates the blinks and has relatively noisy smooth pursuit. Perhaps, this plot can be used for fixations and saccades.

A close up of a map

Description automatically generated

Figure 3 Plots showing Median and SGolay filters on raw unfiltered data

All these are visual observations and by no means to be taken as a fact. As the report progresses, we see that various filters were chosen for each event detection routine. But, from the following observations we have established the following.

Table 2 Summary of observations

|  |  |  |
| --- | --- | --- |
| Filter type | Pros | Use Case |
| Median on confidence filtered | Retains fixations and saccades | Fixation and Saccade detection |
| SGolay on confidence filtered | Retains blinks | Blink detection |
| SGolay then Median | No significant benefit over SGolay only | TBD |
| Median then SGolay | No significant benefit over Median only | TBD |
| Median on unfiltered data | Retains fixations and saccades | Fixations and Saccade detection |
| SGolay on unfiltered data | Retains blinks and crisper smooth pursuit | Blink and smooth pursuit detection |

From Table 2 we deduce that in general, SGolay filtered data can be used to detect Blinks and Smooth Pursuit, and Median filtered data can be used to detect fixations and saccades. In the next section we dive into the algorithm and how each was used in event detection.

## Algorithm

### 6.1. Blink Detection

From Figure 2 (d), we observe that blinks produce certain kind of troughs. To examine this further, Central difference algorithm is applied to the confidence filtered data (Figure 4).

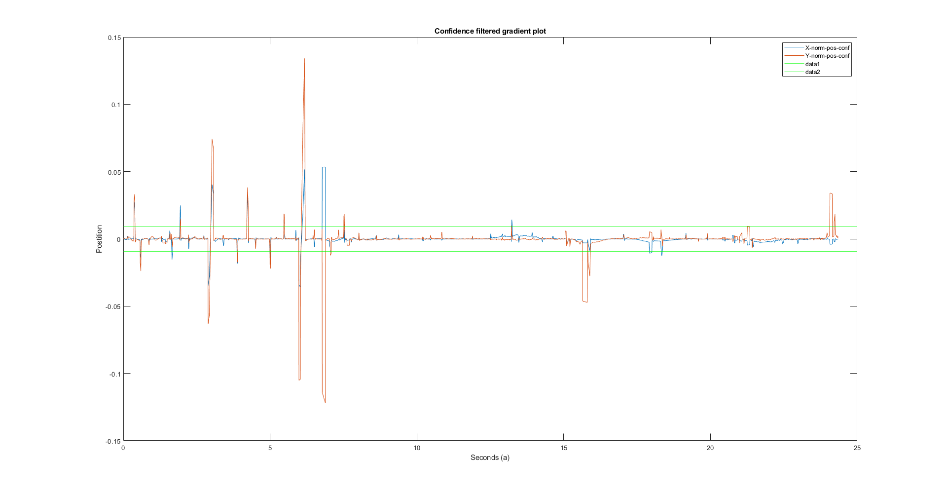


Figure 4 Gradient plot of confidence filtered data with Mean +/- 3\*Std. Dev. in green

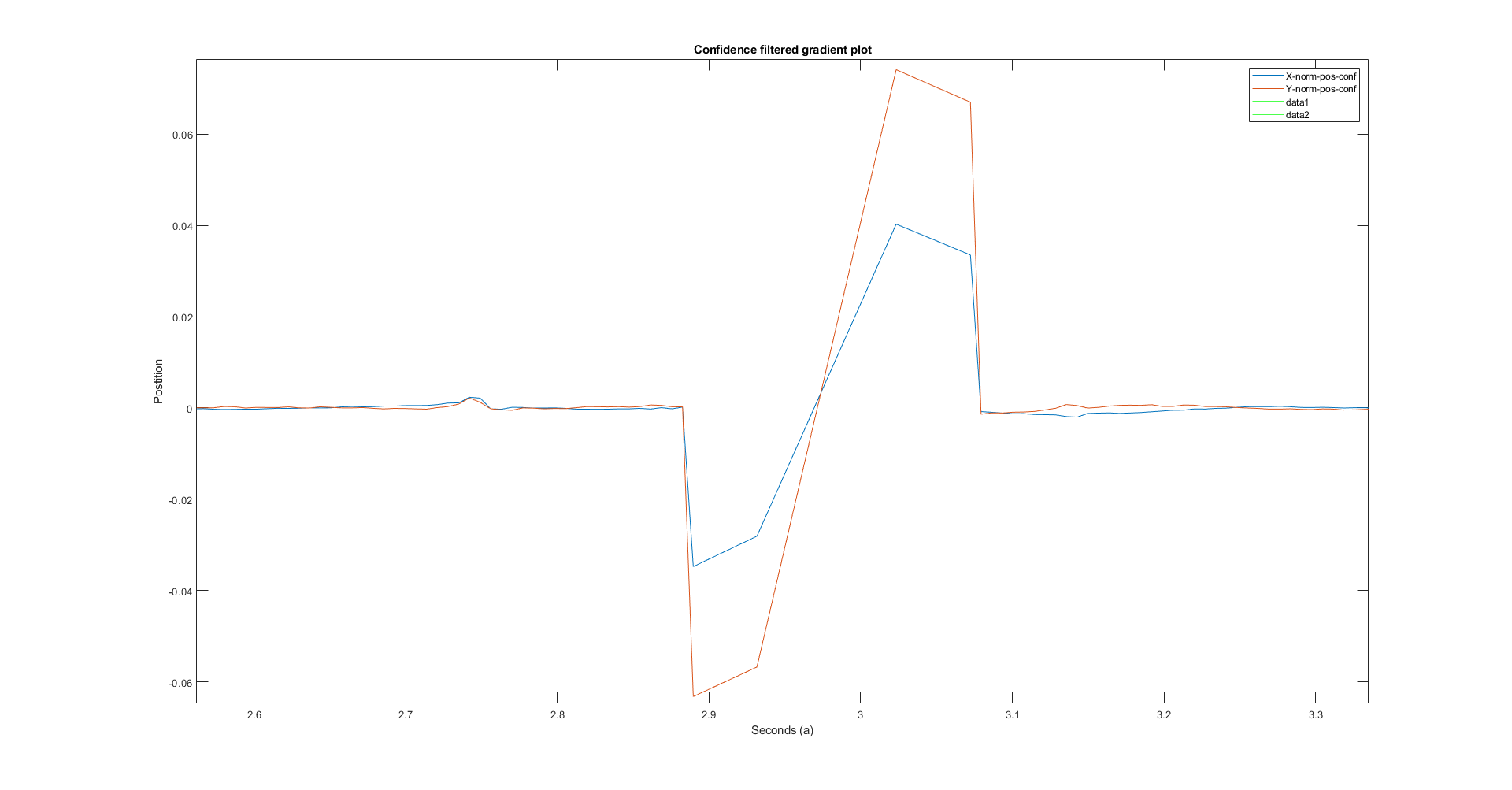


Figure 5 Zoomed in view of highlighted blink in Figure 4

We observe the following:

1. Both blinks, have a steep drop followed by a steep, positive slope jump (Figure 5).
2. Both have extremes beyond Mean +/- 3\* Standard Deviation. (Figure 4)
3. The phenomenon highlighted in black (Figure 4) looks like a blink but the x-y gradients are not in phase

###### Pseudocode

function [change, interval] = blink(vector)

% Declare the array to store last points below -3 \* SD

change = [];

% Declare the array to store points of change of slope direction

interval = [];

% Find the Standard deviation of the vector passed

sd = std(vector);

% first find the negative point below -3 \* SD

for all elements in vector

if (point value is less than -3 \* SD)

insert this value into change

end

end

for all elements in change

% Check if the point after the points stored in change are above 3

% \* SD

if (the next point value is > +3 \* SD)

insert this value into interval

end

end

return interval and change to be plotted

end

As a result, we get the following plots (Figure 6 and Figure 7). We see that both the plots overlap. Hence, the assumption of blinks following this zig-zag pattern is mildly supported. Plotting this with confidence filtered data we get Figure 8. Here we observe that our assumption is further supported. Plotting these detected blinks with raw data we get Figure 9.

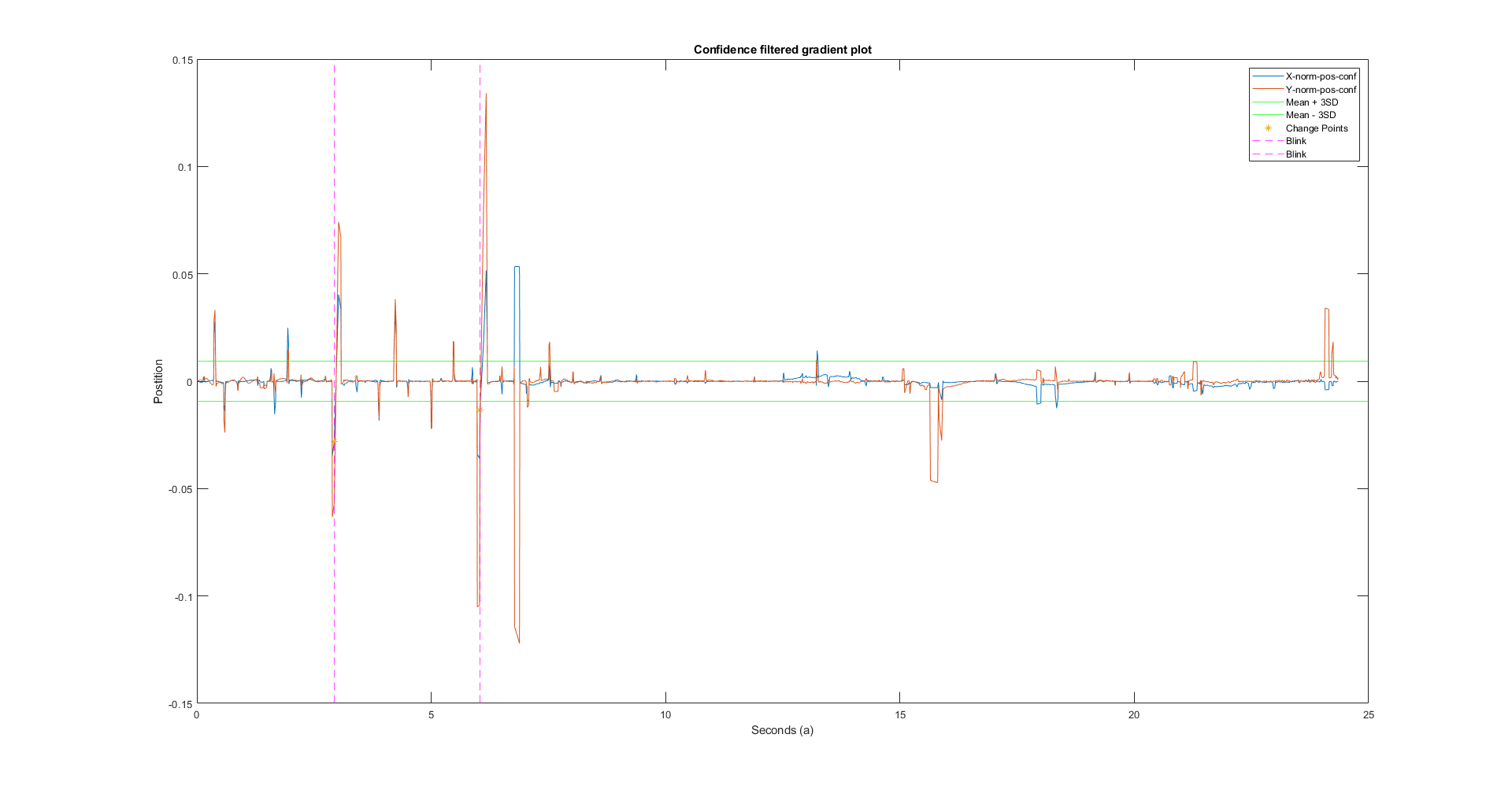


Figure 6 Blinks detected based on X values

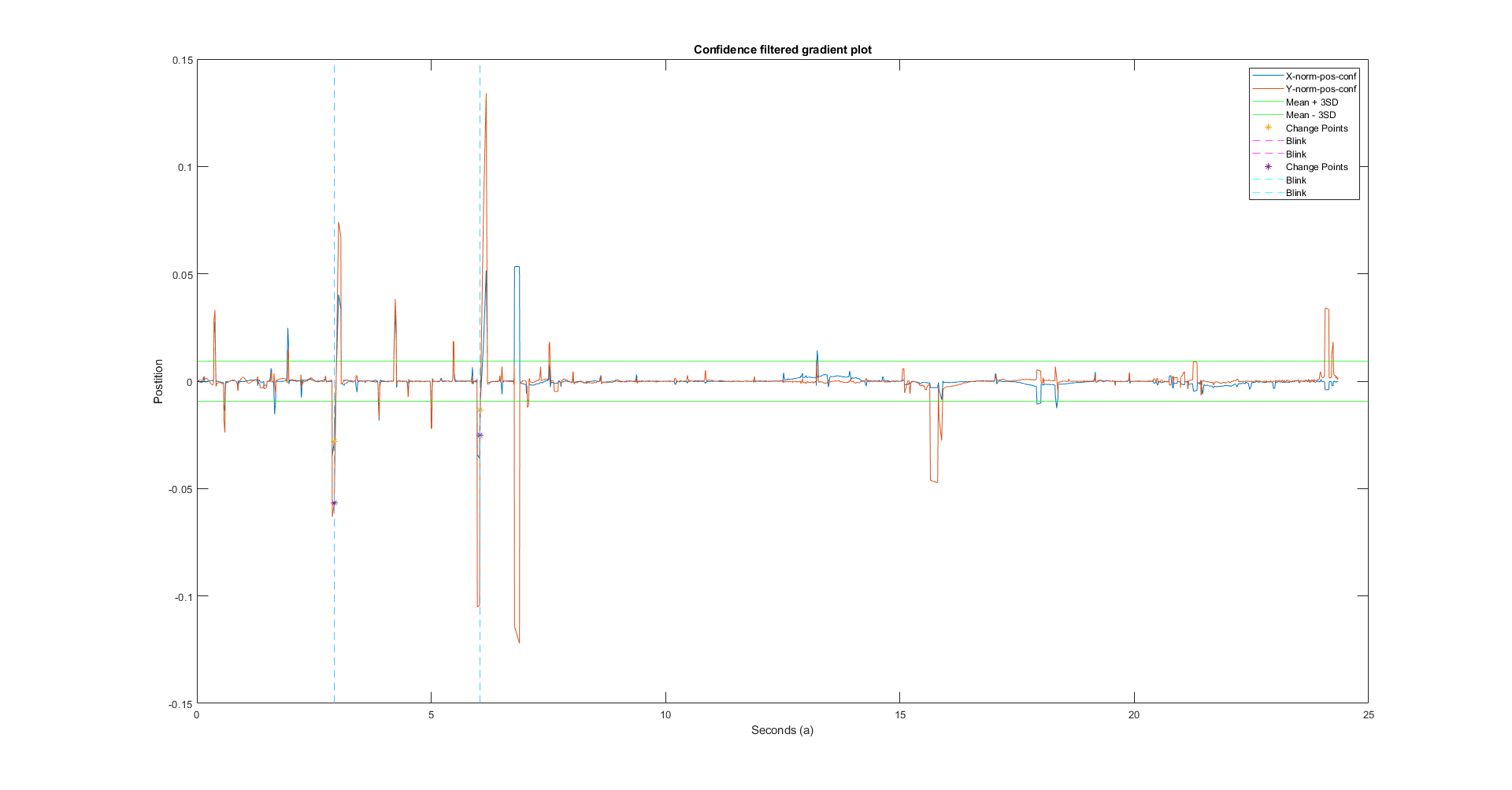


Figure 7 Blinks detected based on Y values

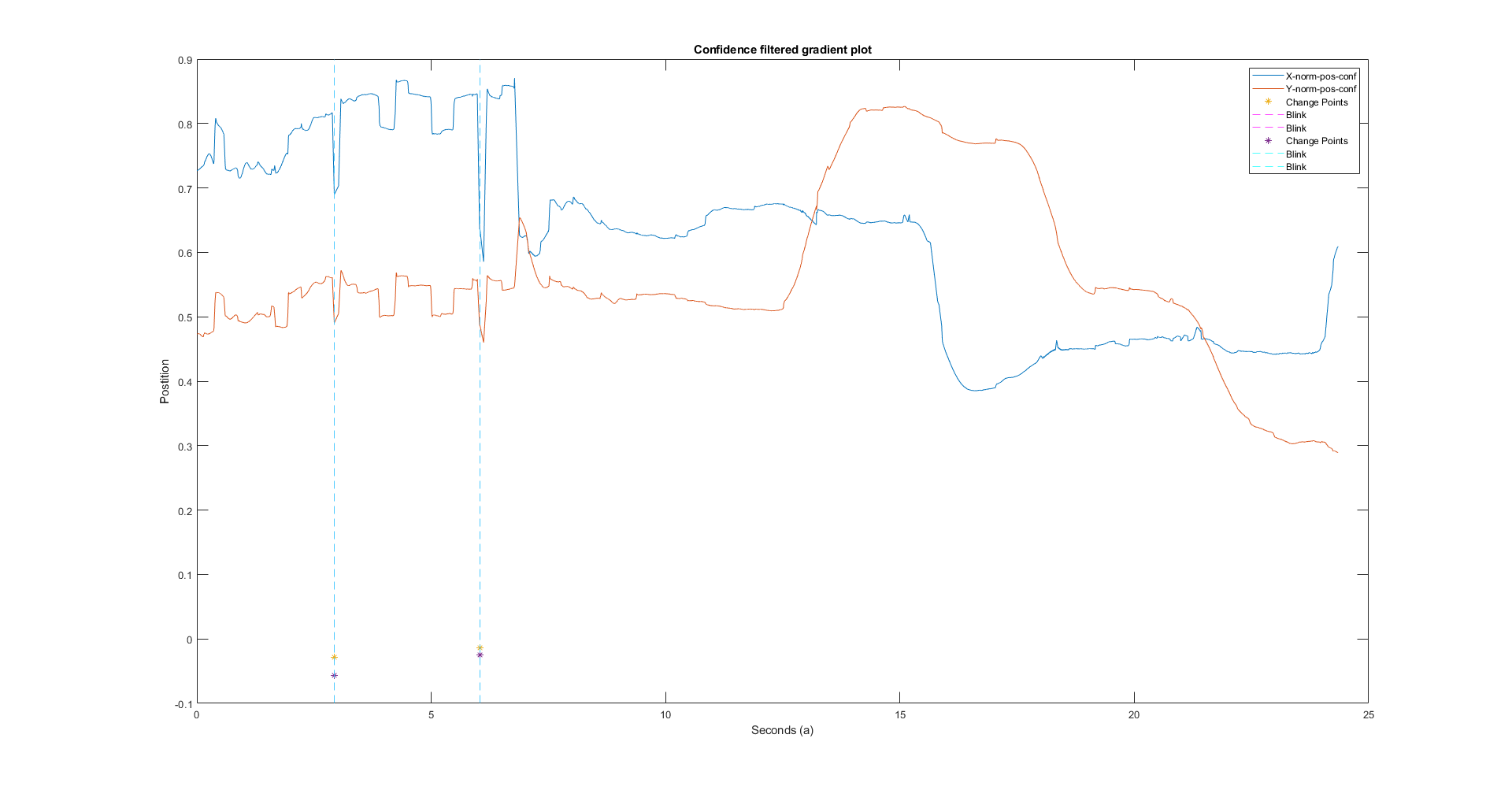


Figure 8 Blinks plotted on confidence filtered data

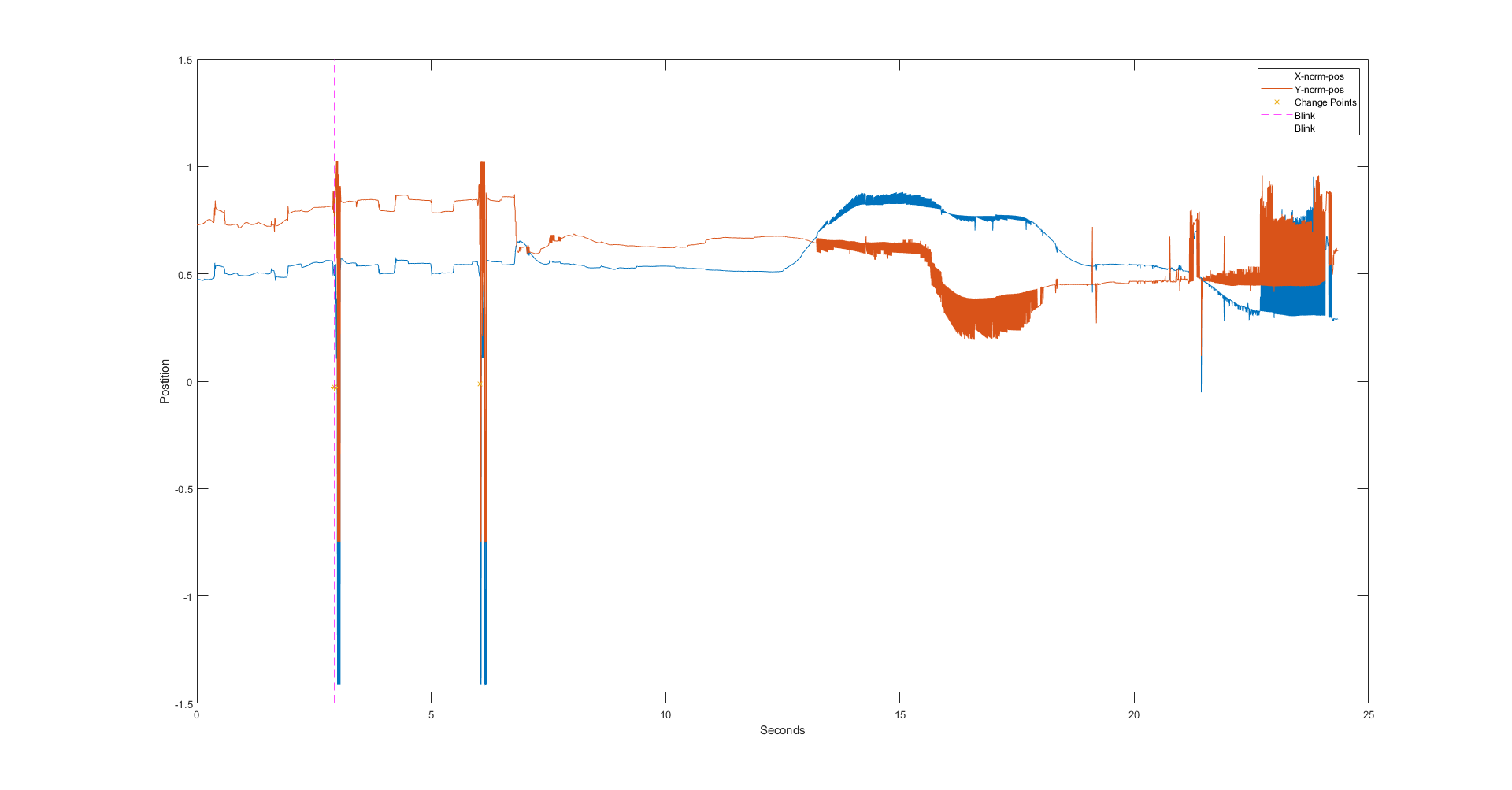


Figure 9 Plotting detecting blinks with raw data

###### Success of the algorithm

This algorithm successfully detects the two blinks present in our 25 seconds of captured data. The biggest benefit of this approach is that the algorithm does not detect the phenomenon highlighted in magenta (Figure 9) as a blink, even though it might look like one.

###### Suggested Improvements

This algorithm is far from perfect. Blinks are a continuous and this algorithm detects a point change in a gradient plot of confidence filtered data. There is partial loss of data due to filtering and inaccuracy in detection due to digital nature of data. Furthermore, In the first blink, the plot is very slightly offset from the actual blink. This suggests that the parameters of the filters need to be adjusted to take the small, rapid changes into account.

### Fixation detection

This algorithm attempts to take a different direction and use clustering algorithms to find fixations. The basic idea behind this algorithm is that if there is a scatter plot of the given data (x-y), then fixations must be *clustered* around a centroid. There can be many centroids. Each centroid represents an estimate of a fixation point. To make it easier, theoretically we can replace the entire cluster with the centroid and have a directed graph constructed using Kruskal’s algorithm. For this clustering, we choose DBSCAN algorithm.

DBSCAN attempts to find clusters around every point in the data. This cluster has a radius of Ɛ (epsilon, a predefined value), which in this case is 2 degrees and the minimum number of points required to qualify as a cluster is 10. The following (Figure 10) is the initial scatter plot of time-y-x data.

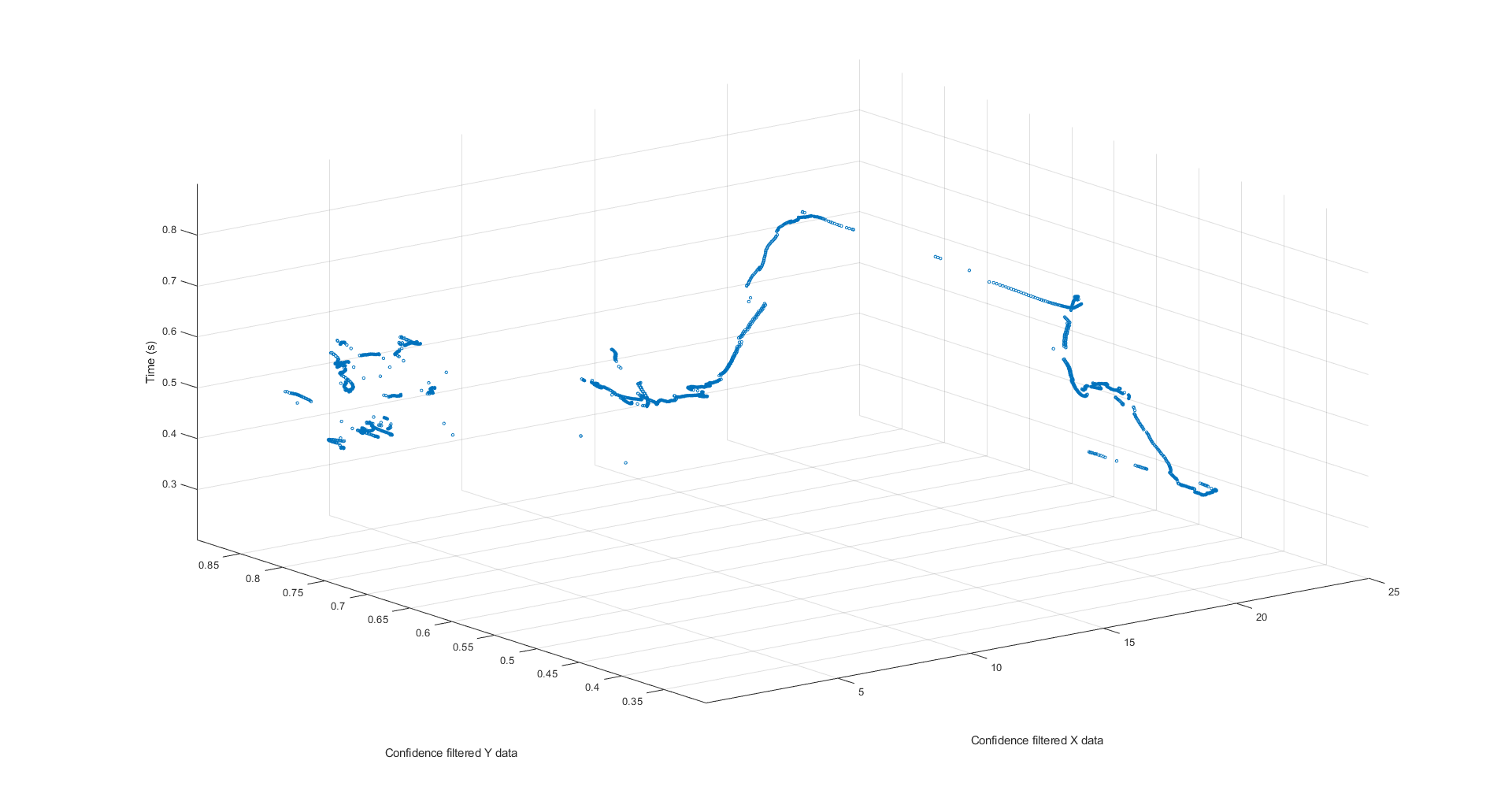


Figure 10 3D scatter of time-y-x data

Unfortunately, MATLAB does not have DBSCAN built in and referring and mixing-matching from various sources [1, 2, 3] did not help as they have issues of their own. Therefore, this algorithm was shelved.

Another attempt to cluster the data was done using the Hierarchical Clustering algorithm [4].

###### Pseudocode:

Start with X-data

Step 1: Find the distances (pdist in MATLAB) between all given points

Step 2: Construct an Agglomerative Hierarchical Cluster Tree (linkage in MATLAB [5])

Step 3: Run the dissimilarity check for the tree constructed in step 2 by computing the Cophenetic Correlation Coefficient (cophenet in MATLAB [6]) of the Tree (Step 2) and the distance matrix

Step 4: Run the dissimilarity check again but this time with the distance calculated using the ‘cityblock’ filter and the tree constructed using the ‘average’ parameter.

Step 5: Verify the inconsistency of the original tree (Step 2)

Step 6: Create the clusters using the tree in Step 2 as input and cutoff value of 1 for thresholding.

Step 7: Cross reference the cluster number assigned to each entry in Step 6 with the original data and find the mean of every centroid

Step 8: Repeat this step for Y-data

Step 9: Plot all found centroid of the clusters.

The dendrogram plot of the tree in Step 2 looks as shown below in

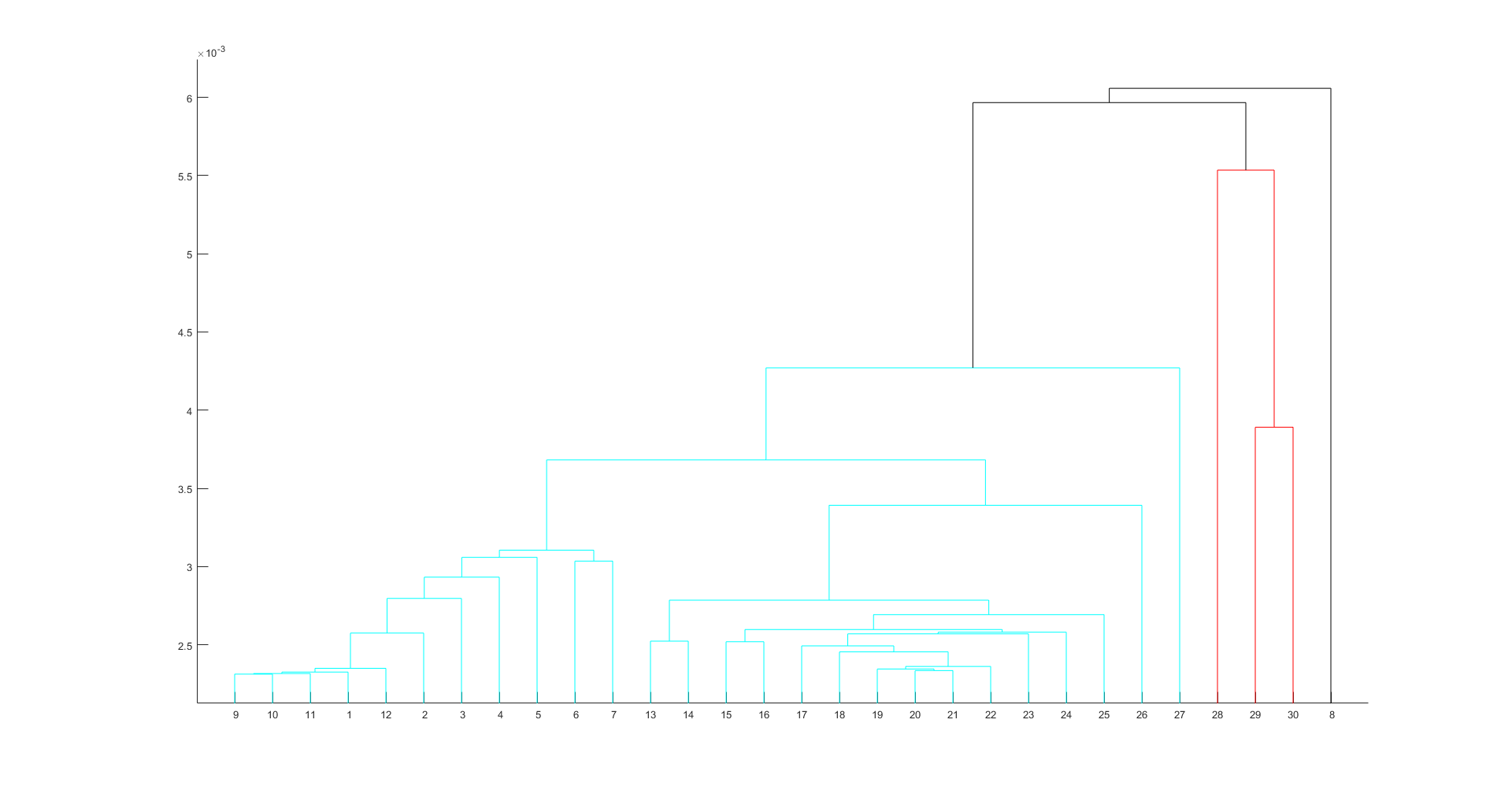


Figure 11 Dendrogram of the constructed Agglomerative Hierarchical Cluster Tree

The colors show the how the clustering (only for visual purpose) would take place. Currently, due to much of time spent on attempts to incorporate DBSCAN in MATLAB, this code is incomplete.

###### Success of the algorithm

TBD

###### Suggested Improvements

This algorithm does not include temporal data. This results in clustering points that occur at different points in temporal space. One way to account for time would be to cluster in a 3D space (Figure 10).

## References:

1. <https://github.com/sinjax/dbscan/blob/master/dbscan.m>
2. <https://www.mathworks.com/matlabcentral/fileexchange/52905-dbscan-clustering-algorithm>
3. <http://yarpiz.com/255/ypml110-dbscan-clustering>
4. <https://www.mathworks.com/help/stats/hierarchical-clustering.html>
5. <https://www.mathworks.com/help/stats/linkage.html>
6. https://www.mathworks.com/help/stats/cophenet.html