Engineering Update on the project: ***"EEG Measures for Schizophrenia"***

April 3, 2023

1 Summary

MMN computation has been altered to be referenced to the standard tone, include min-max scaling and use of a five point moving average for plotting. This improved visual interpretation of the waveforms. The figures are shown in section 3.

Fuzzy entropy computation now add extras space dimensions whose signal is a gaussian noise of a unit standard deviation. So there is no need for combination of electrode from corticla regions to compute fuzzy entropy anymore.

A fuzzy entropy algorithm for a univariate time-series has been implemented and tested. The algorithm is to be improved by being extended for a multivariate time -series(2D data). This algorithm is shown in 4.

Further data has been acquired from 20 more subjects. This data is not included yet. The new data recordings will be included in future processing and reports.

An handheld annotator is being developed for easy feedback to DAQ system from patient and clinician.

2 Next Steps

Over the period of four weeks, I will be doing the following

- Continued development of hand held annotator device.
- Computing MMN amplitudes from the MMN waveforms.
- Improving self-developed fuzzy entropy library to work with multivariate time-series(2D data).
- Recomputing fuzzy entropy features.
- Comparing fuzzy entropy features from developed library to sourced library.
- Computing auditory steady state response and features.

3 Figures

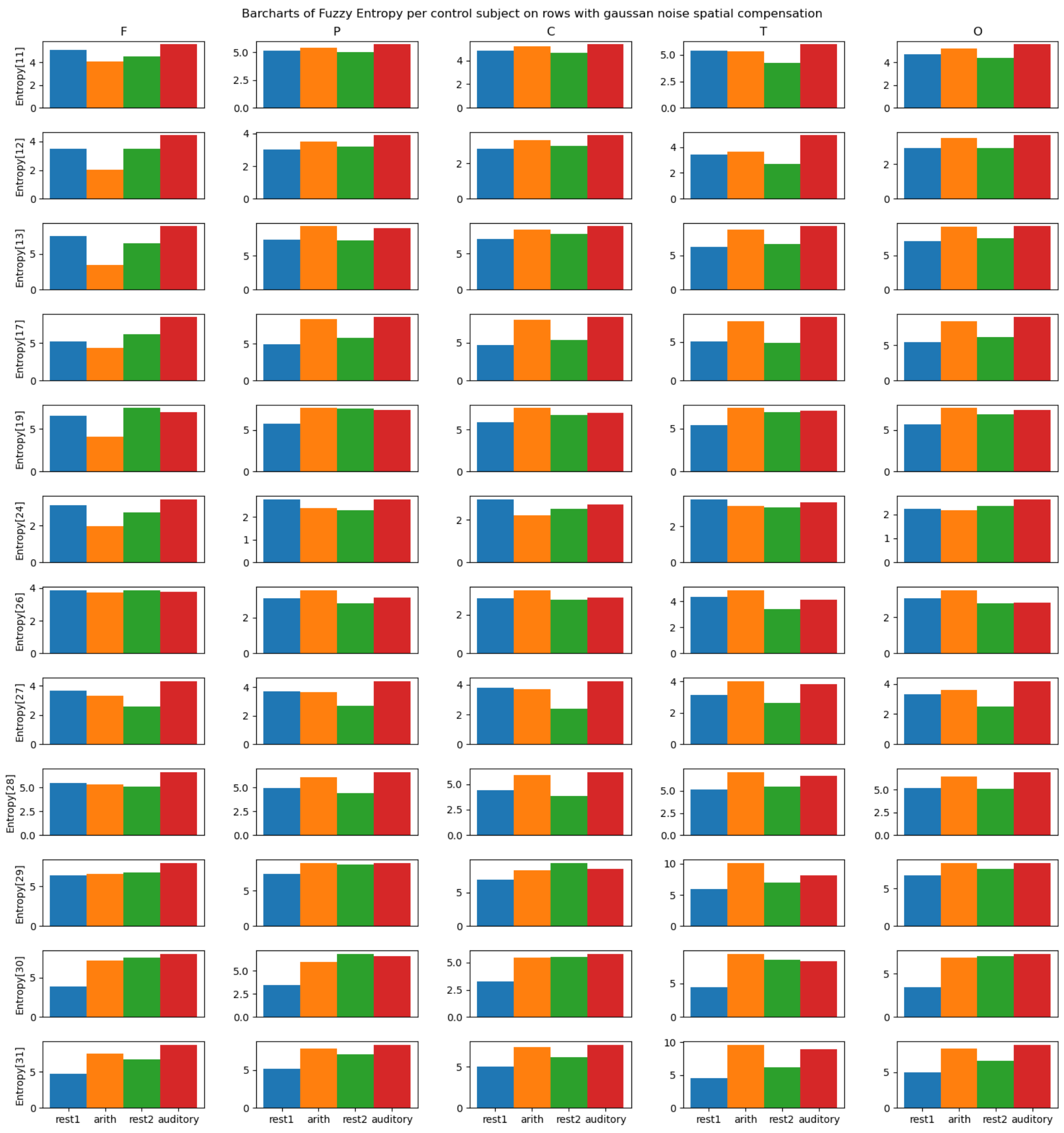

Figure 1: Fuzzy Entropy from controls

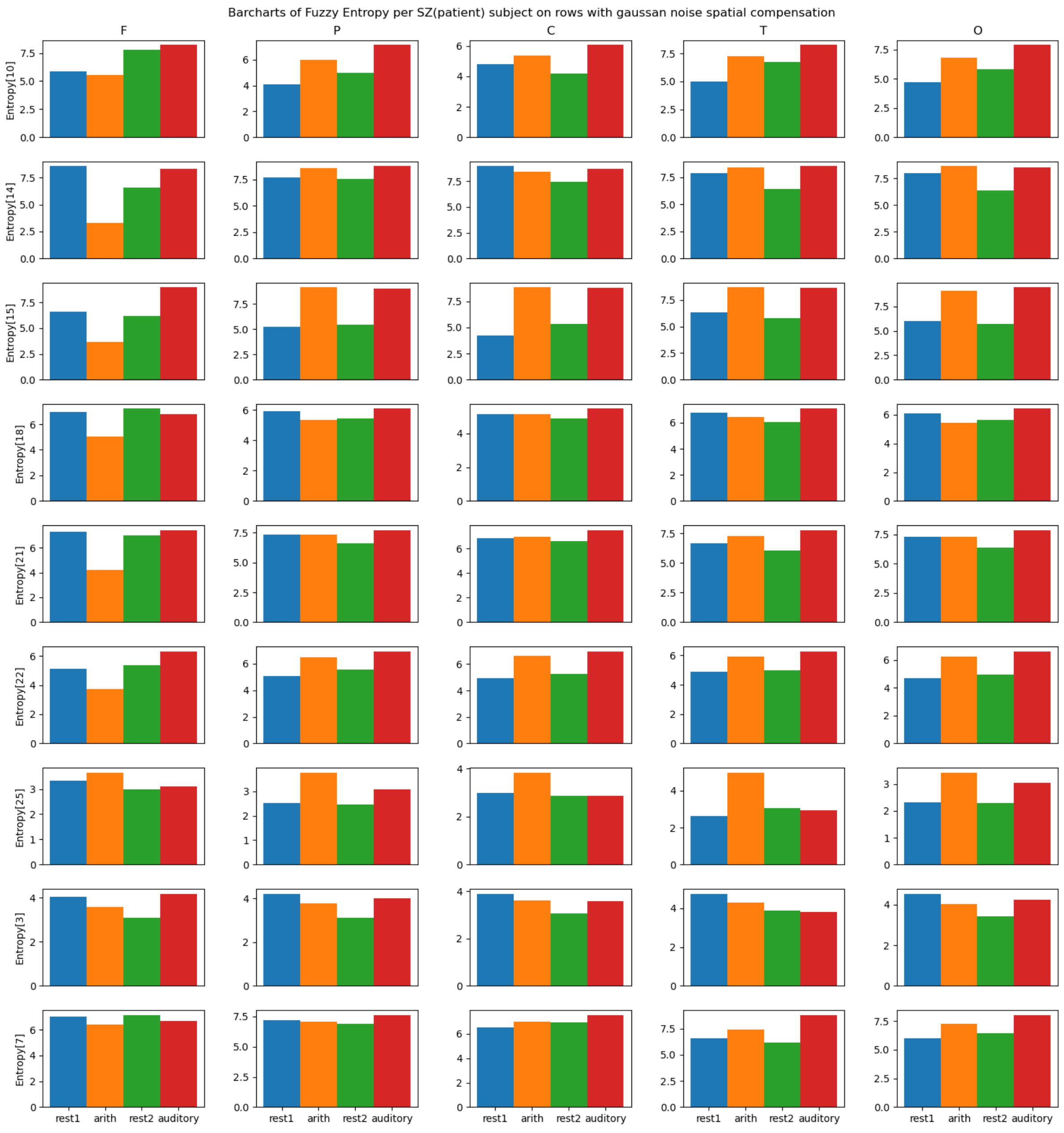

Figure 2: Fuzzy Entropy from patients

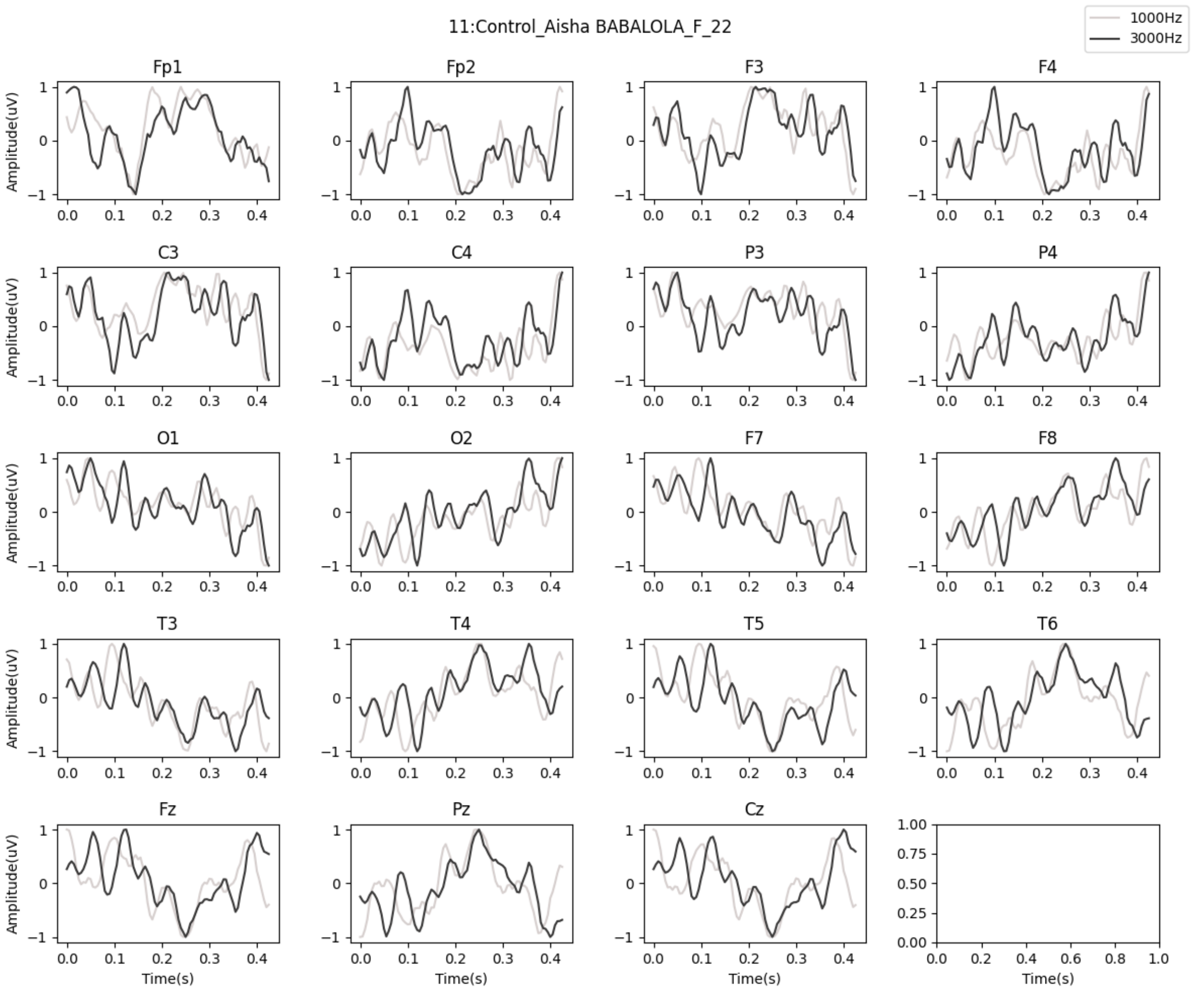

Figure 3: A control subject MMN plots

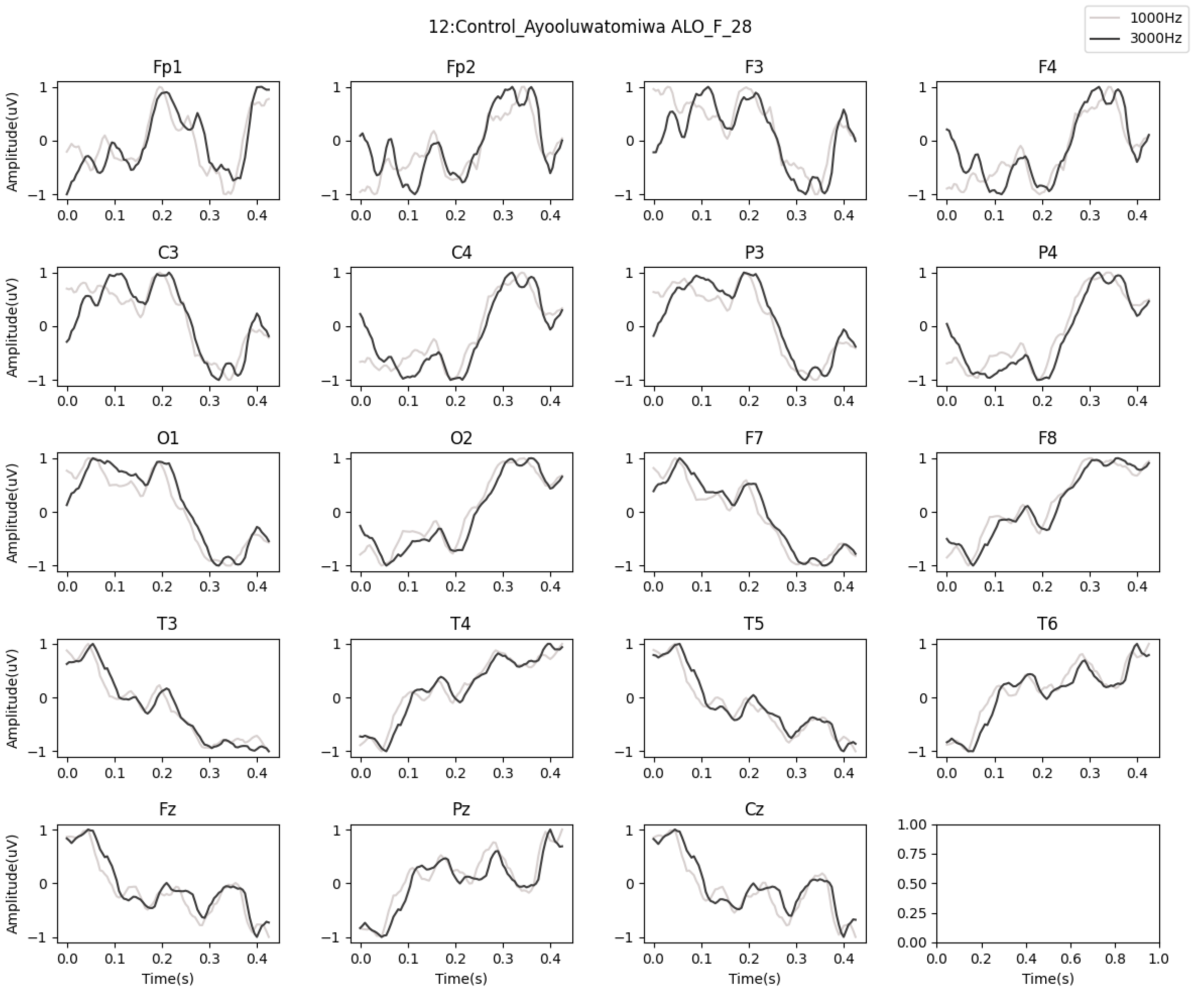

Figure 4: A control subject MMN plots

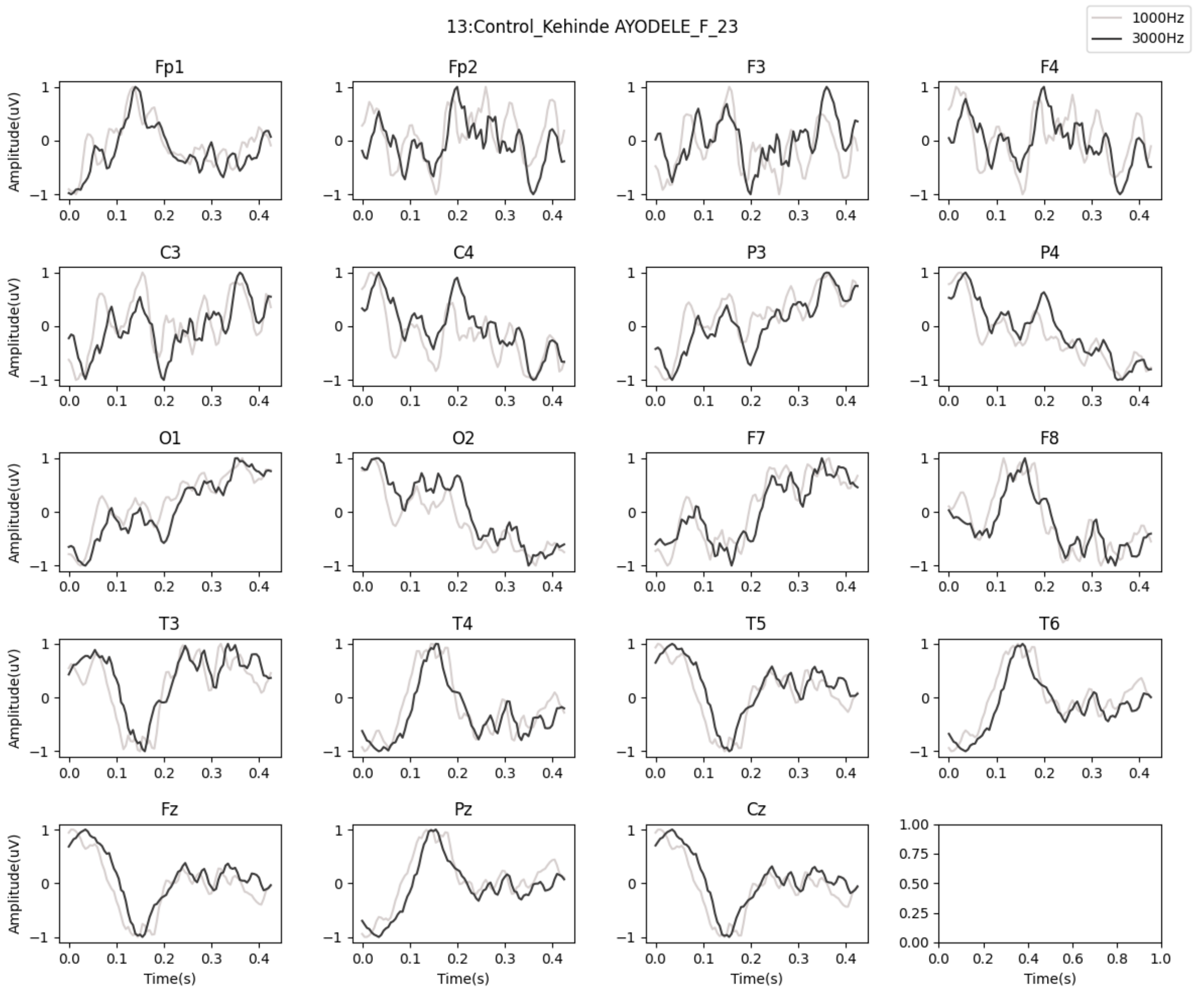

Figure 5: A control subject MMN plots

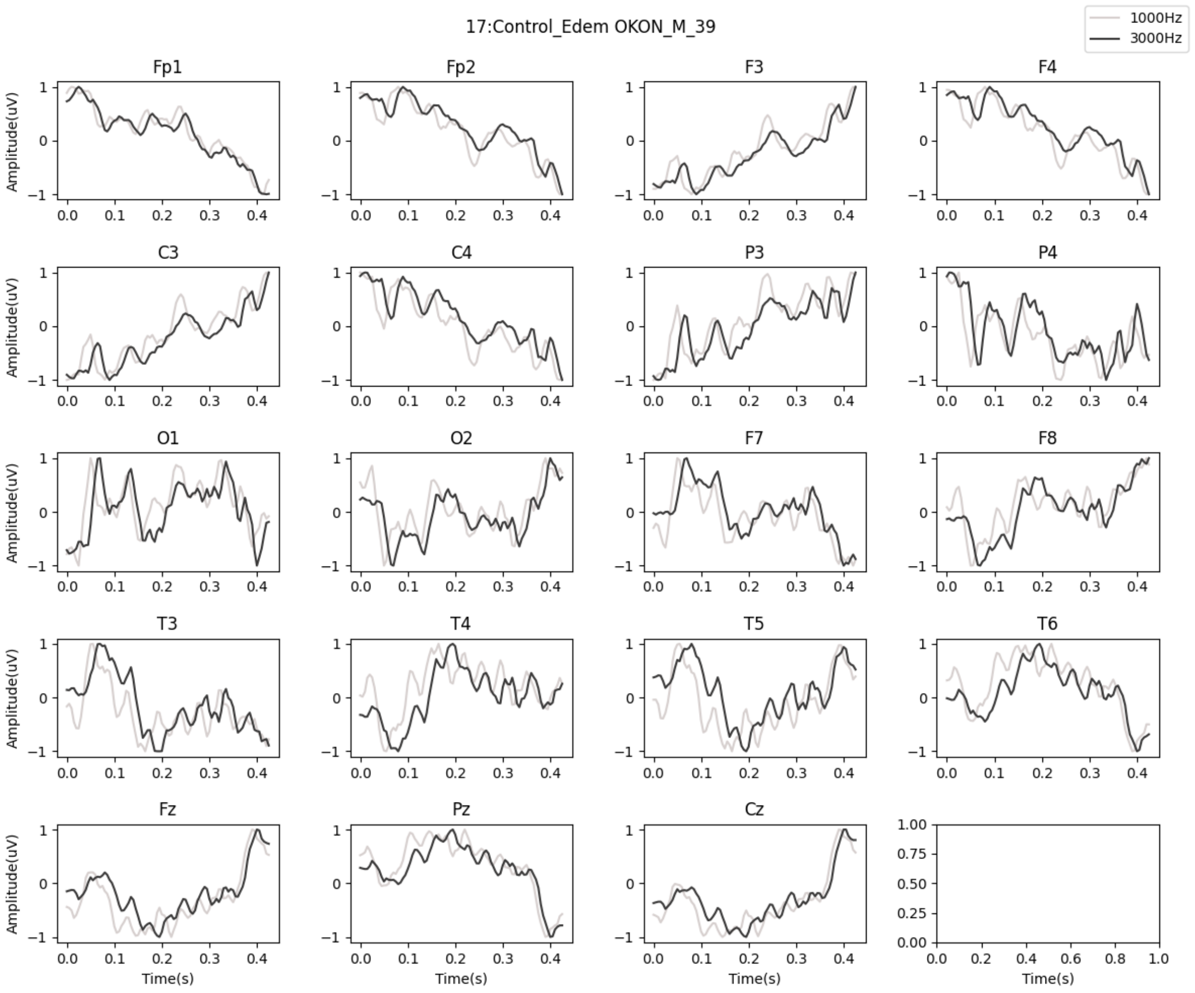

Figure 6: A control subject MMN plots

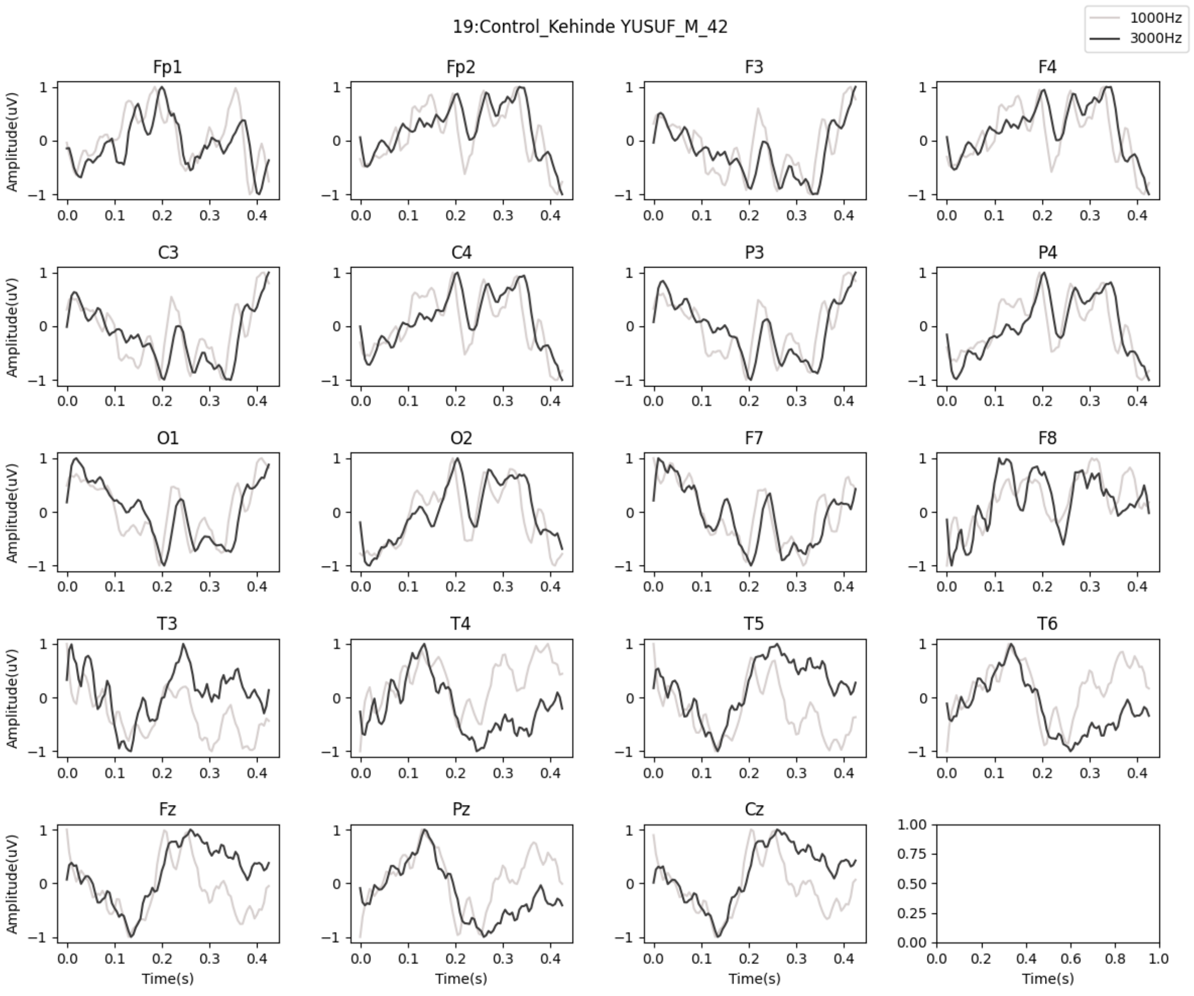

Figure 7: A control subject MMN plots

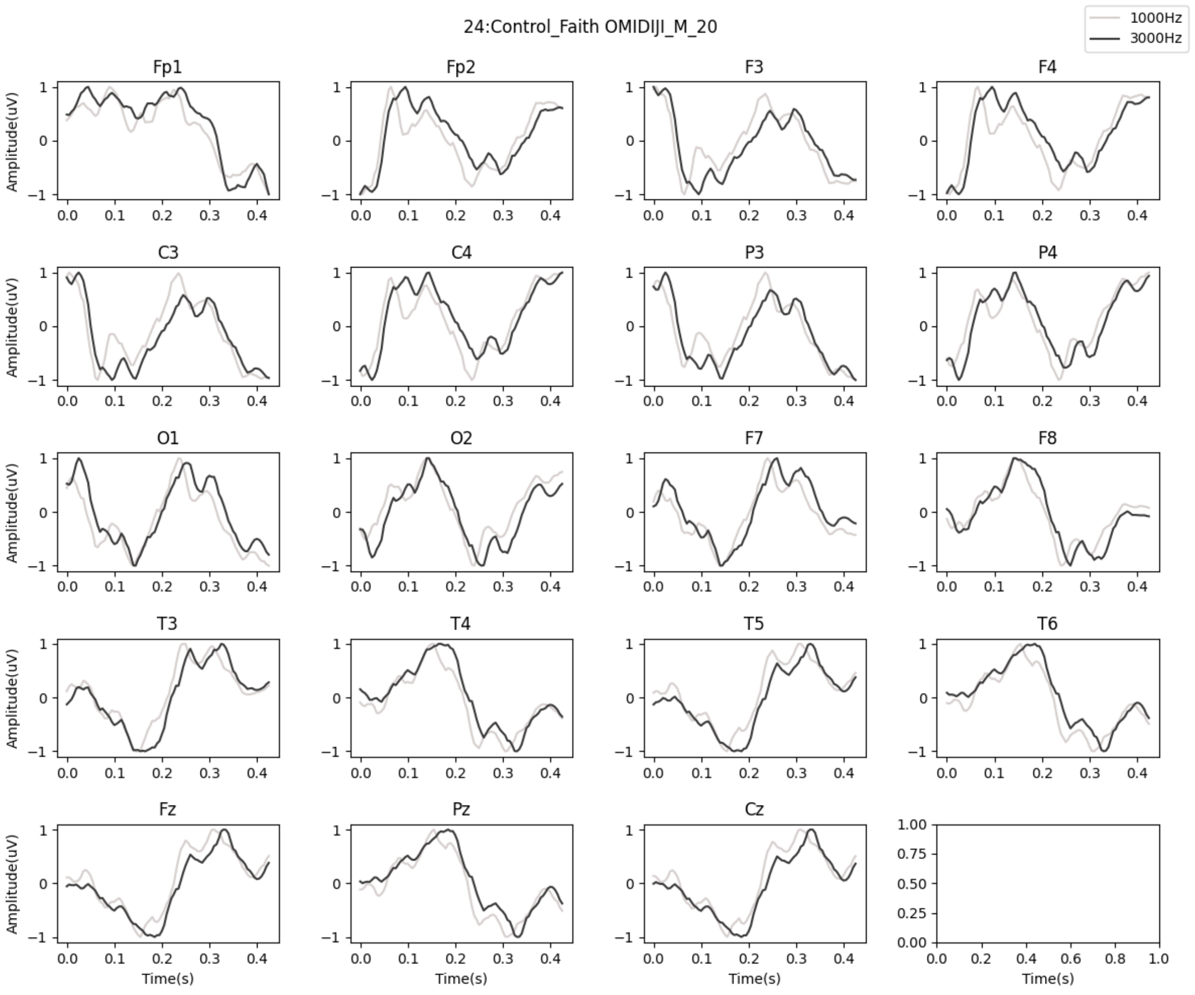

Figure 8: A control subject MMN plots

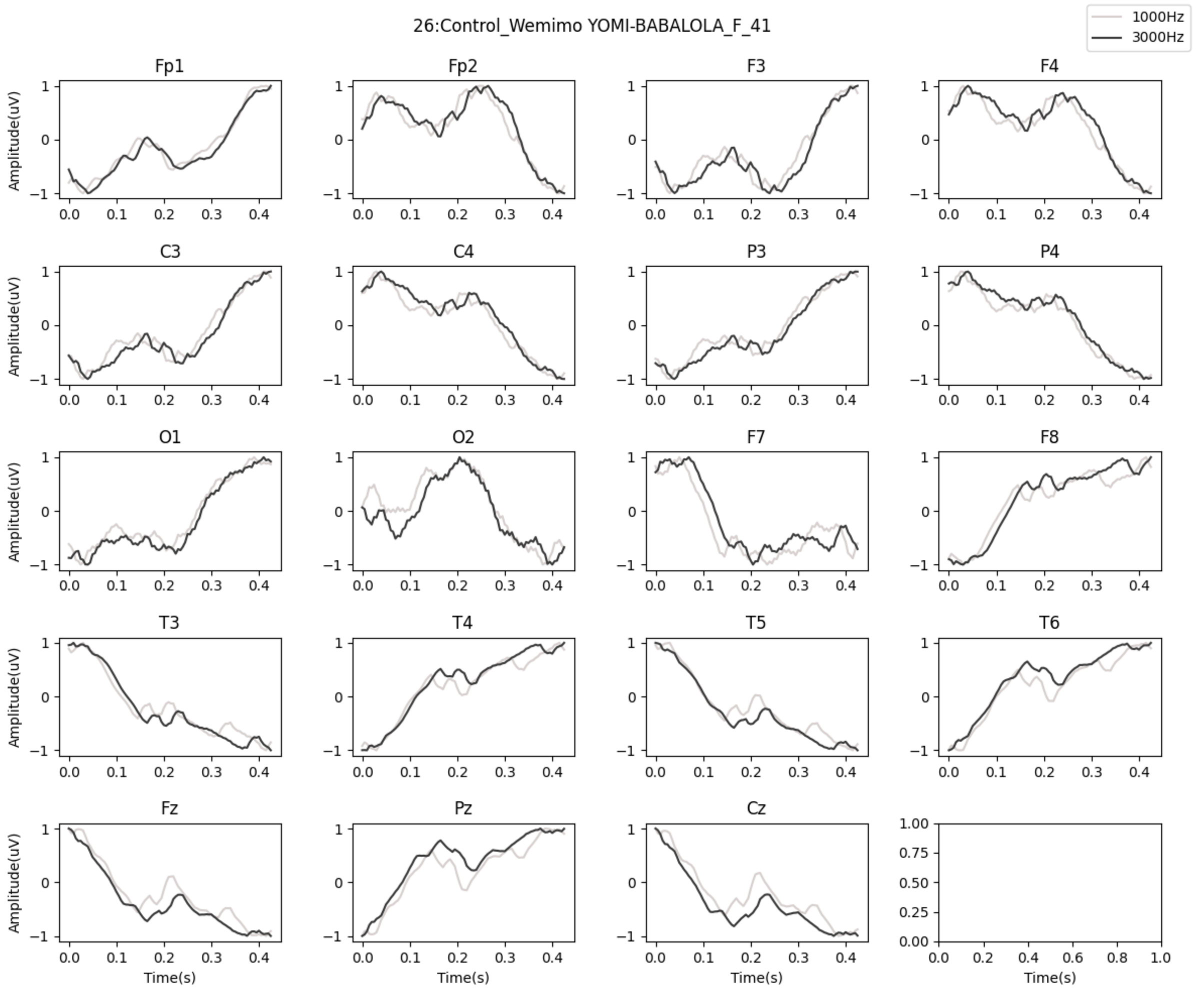

Figure 9: A control subject MMN plots

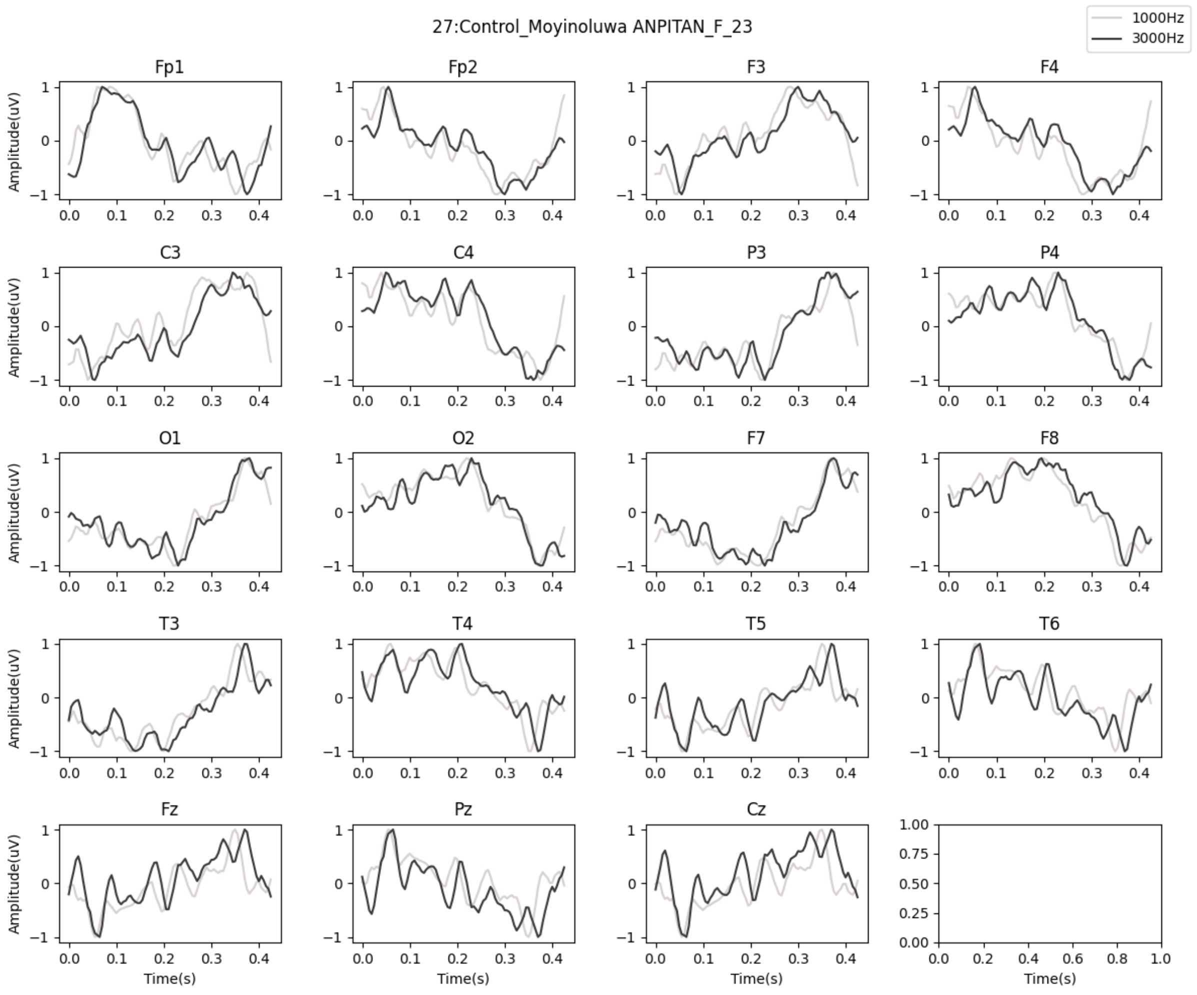

Figure 10: A control subject MMN plots

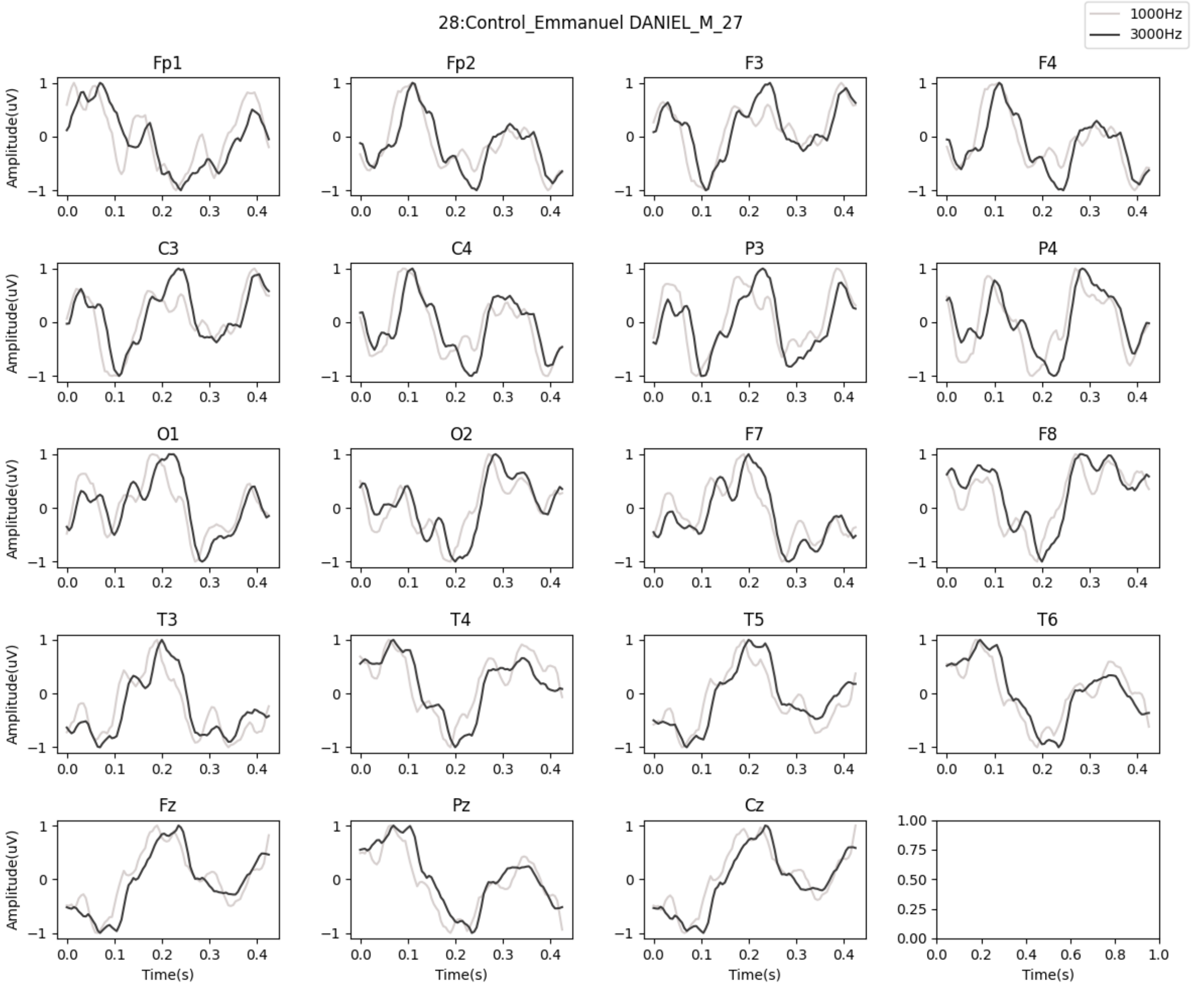

Figure 11: A control subject MMN plots

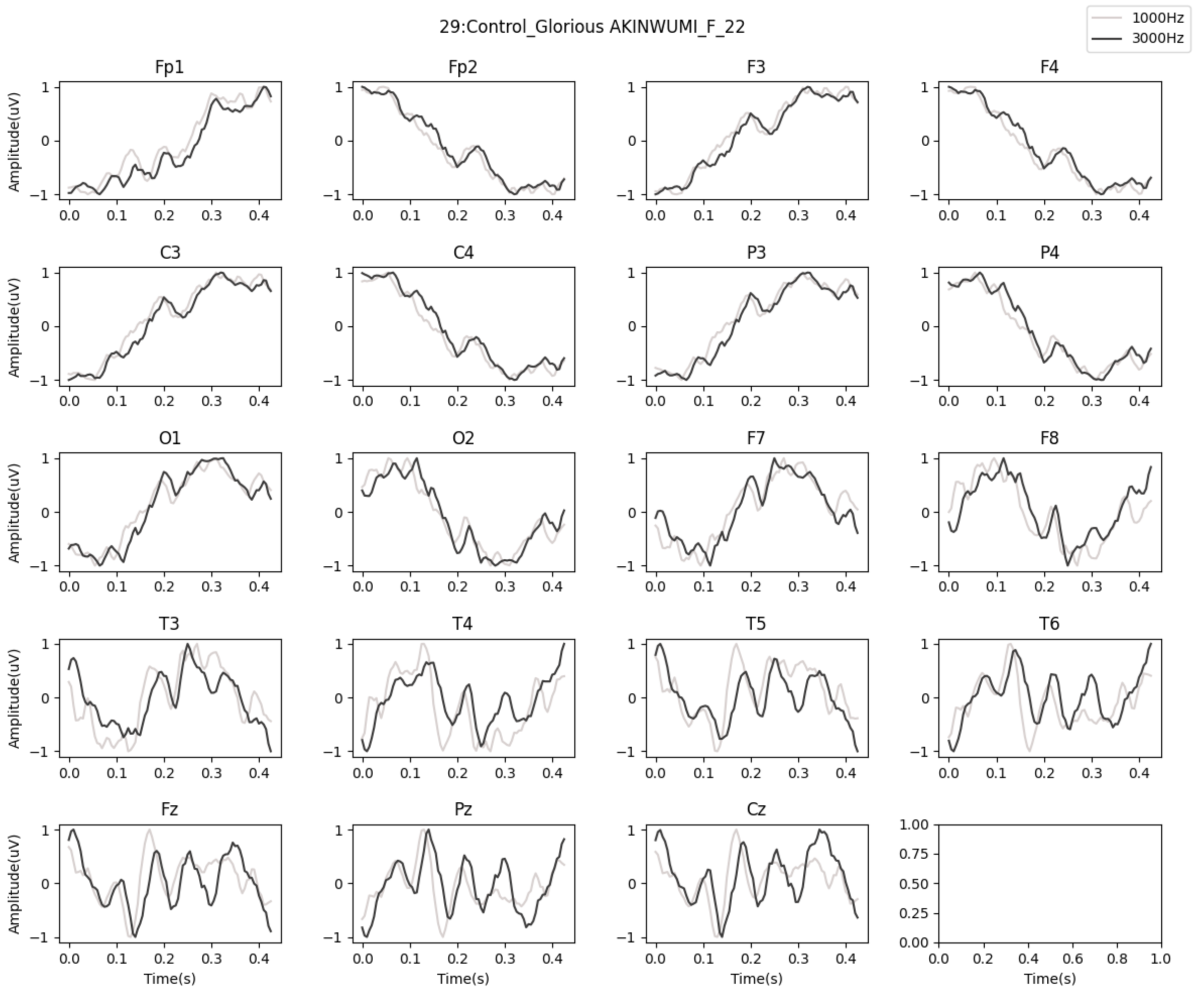

Figure 12: A control subject MMN plots

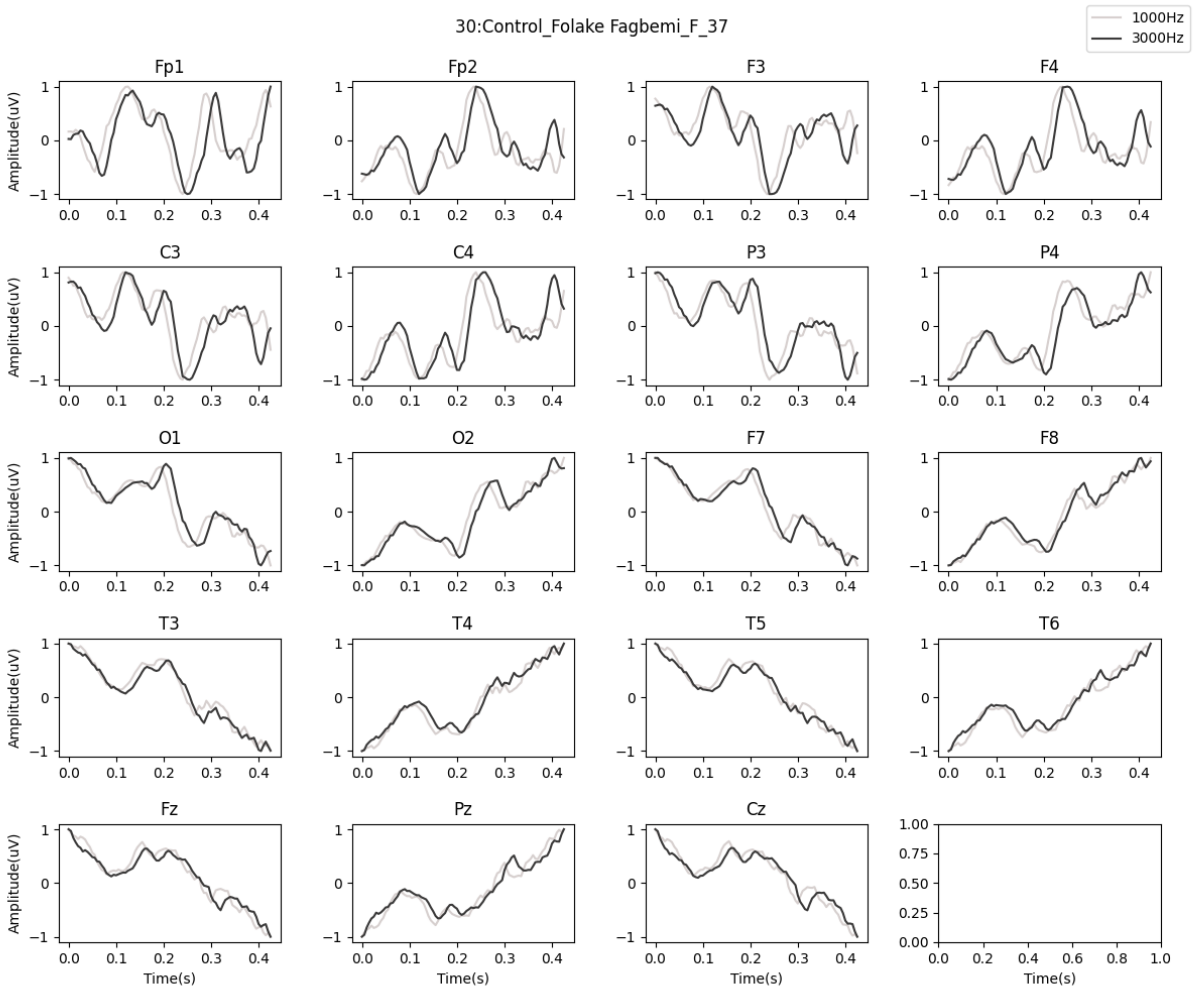

Figure 13: A control subject MMN plots

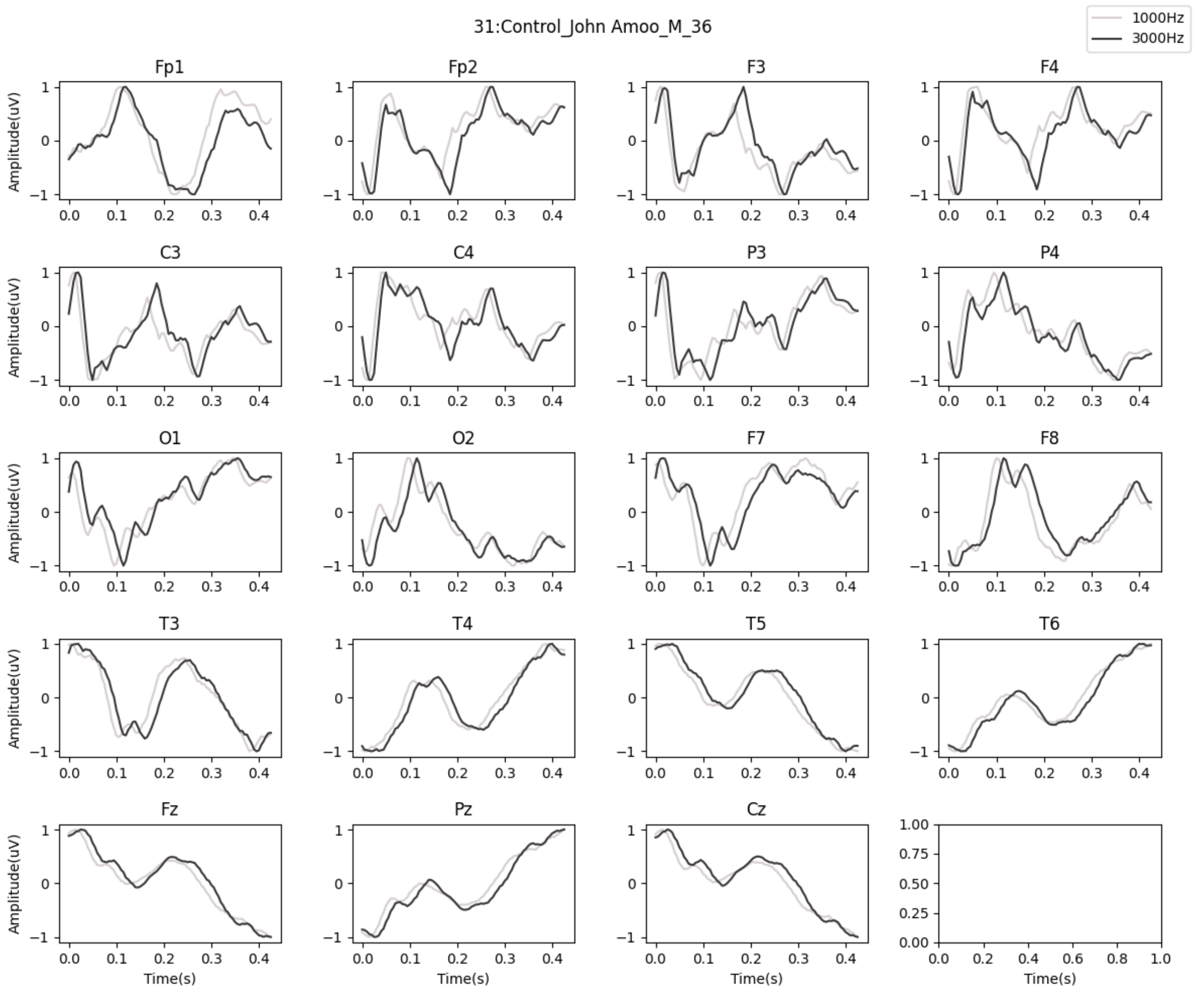

Figure 14: A control subject MMN plots

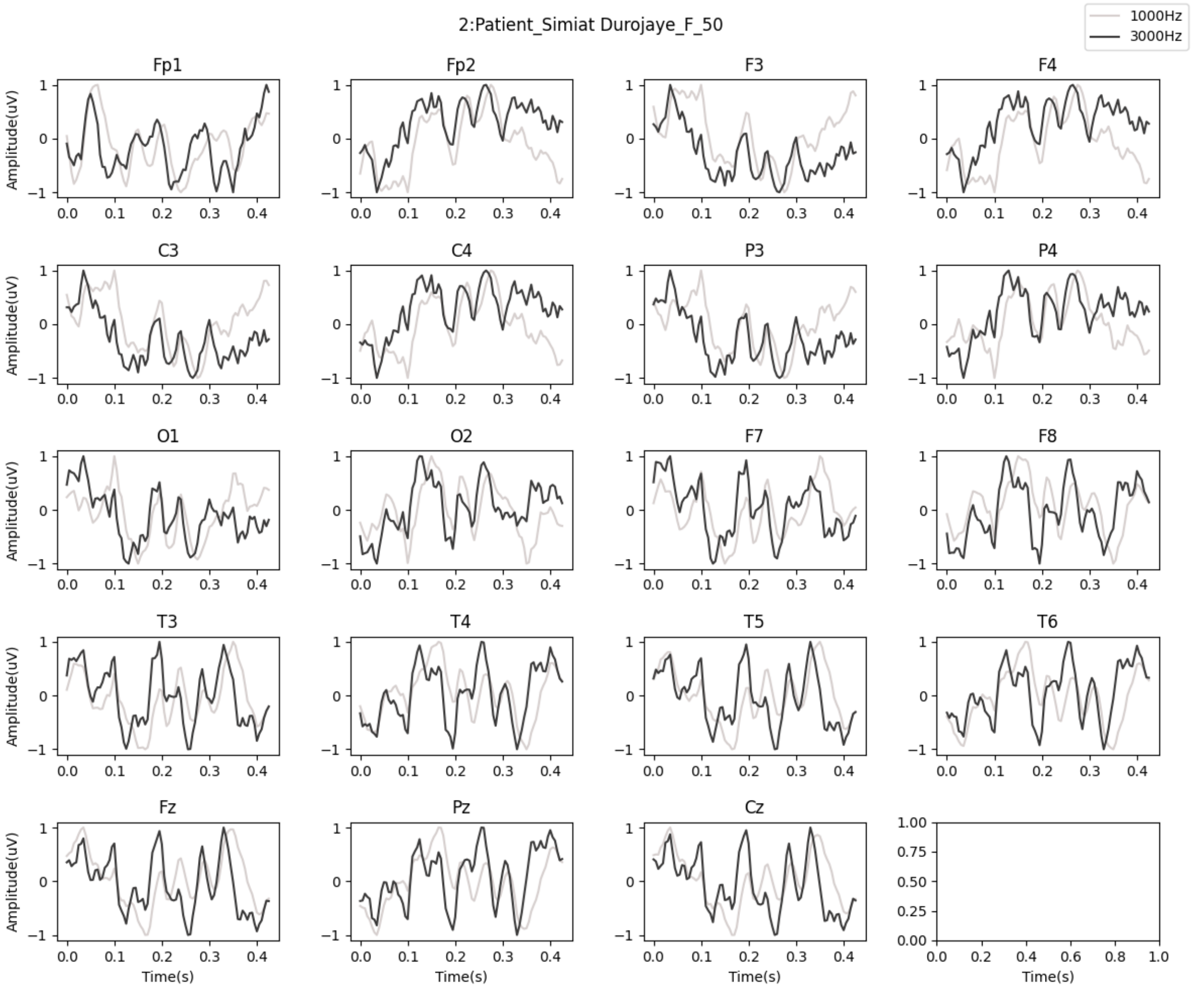

Figure 15: A SZ subject MMN plots

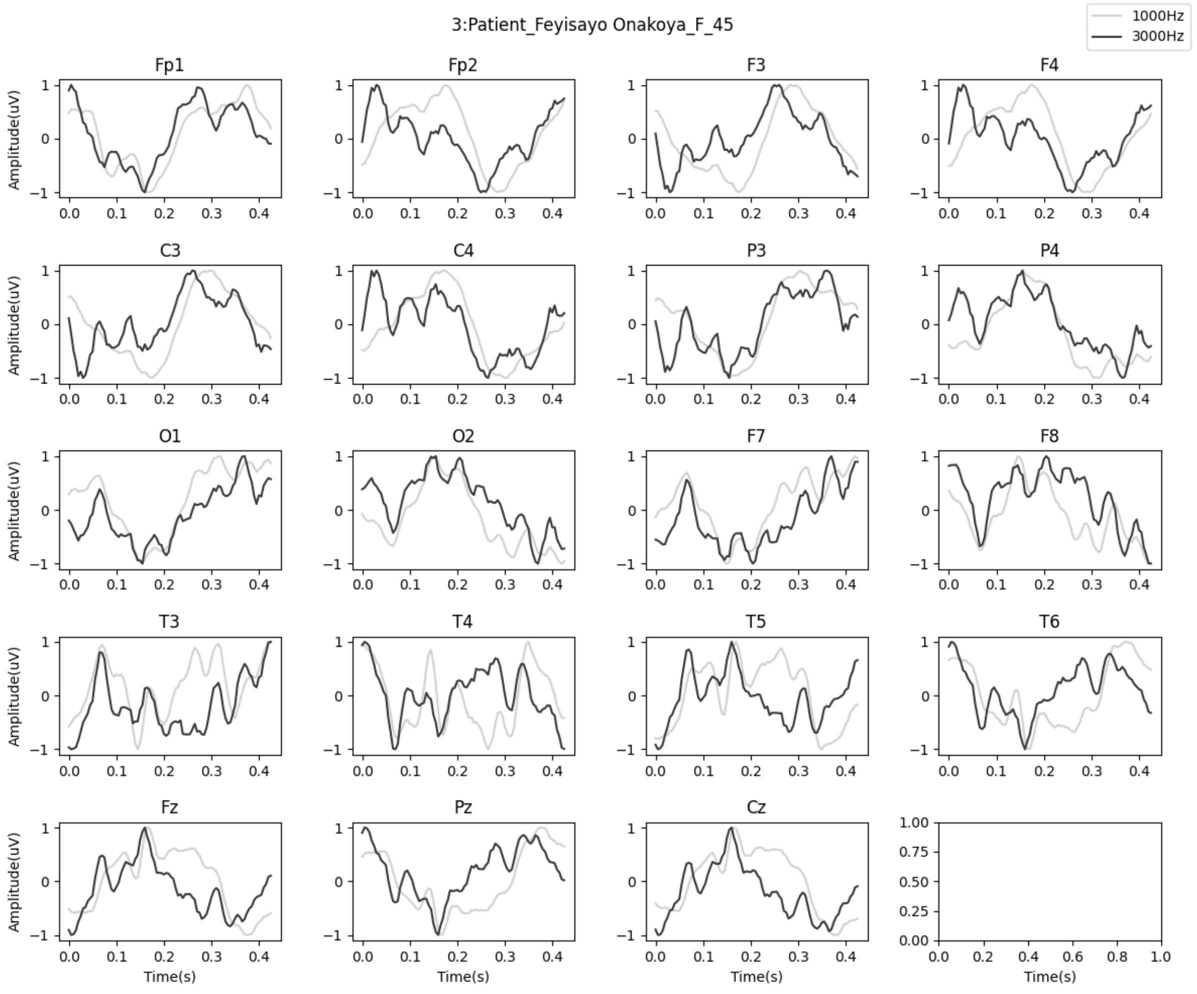

Figure 16: A SZ subject MMN plots

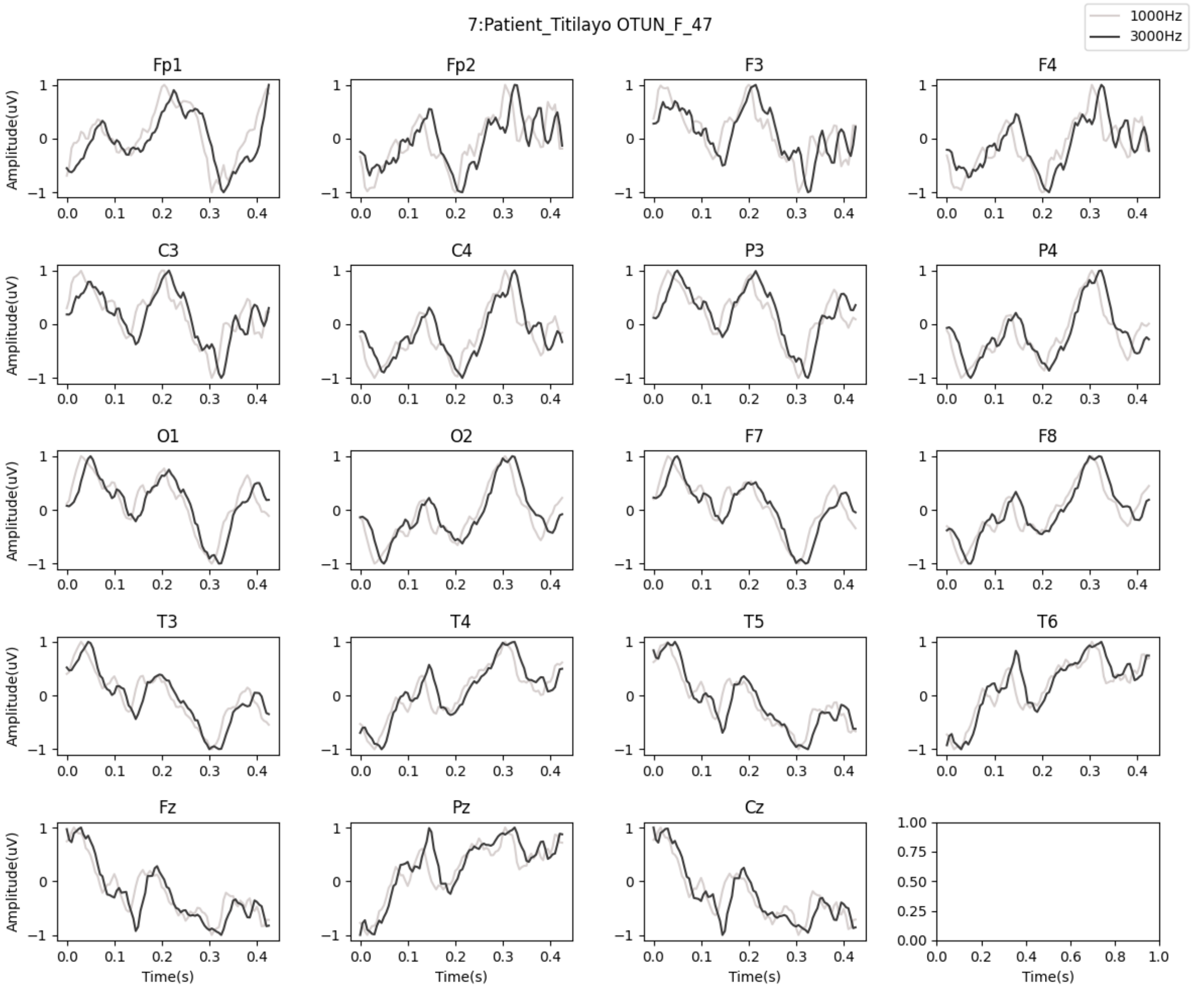

Figure 17: A SZ subject MMN plots

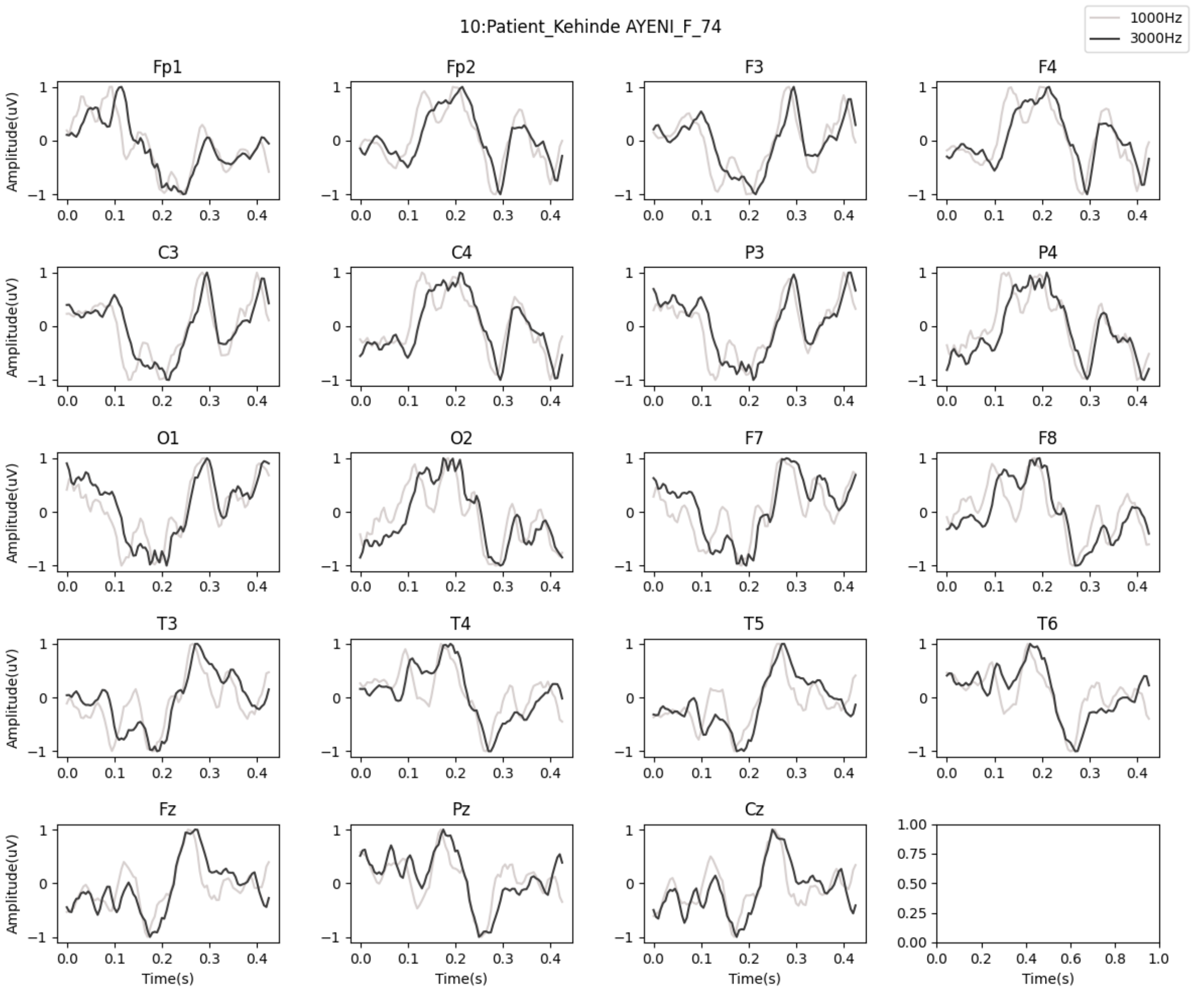

Figure 18: A SZ subject MMN plots

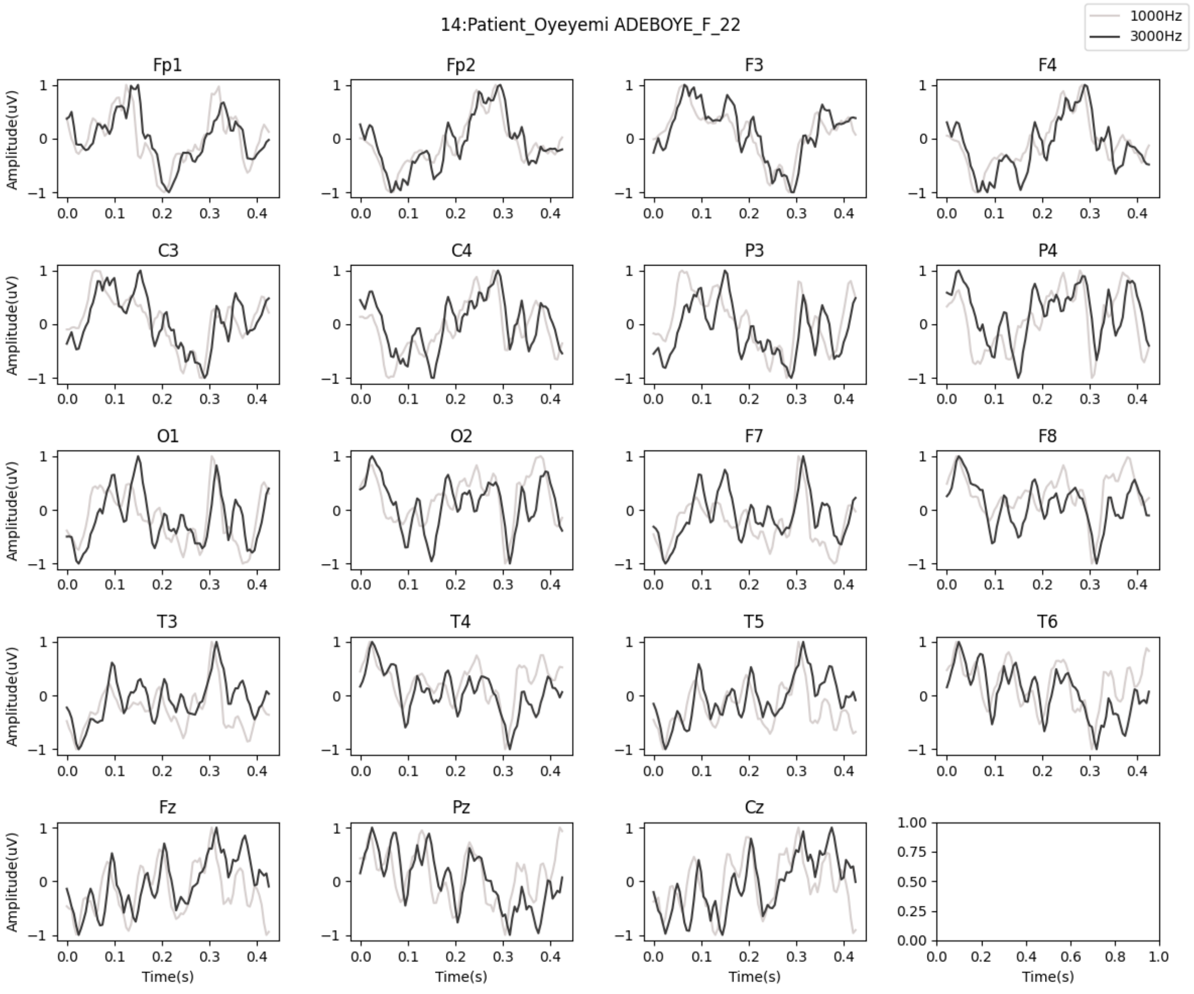

Figure 19: A SZ subject MMN plots

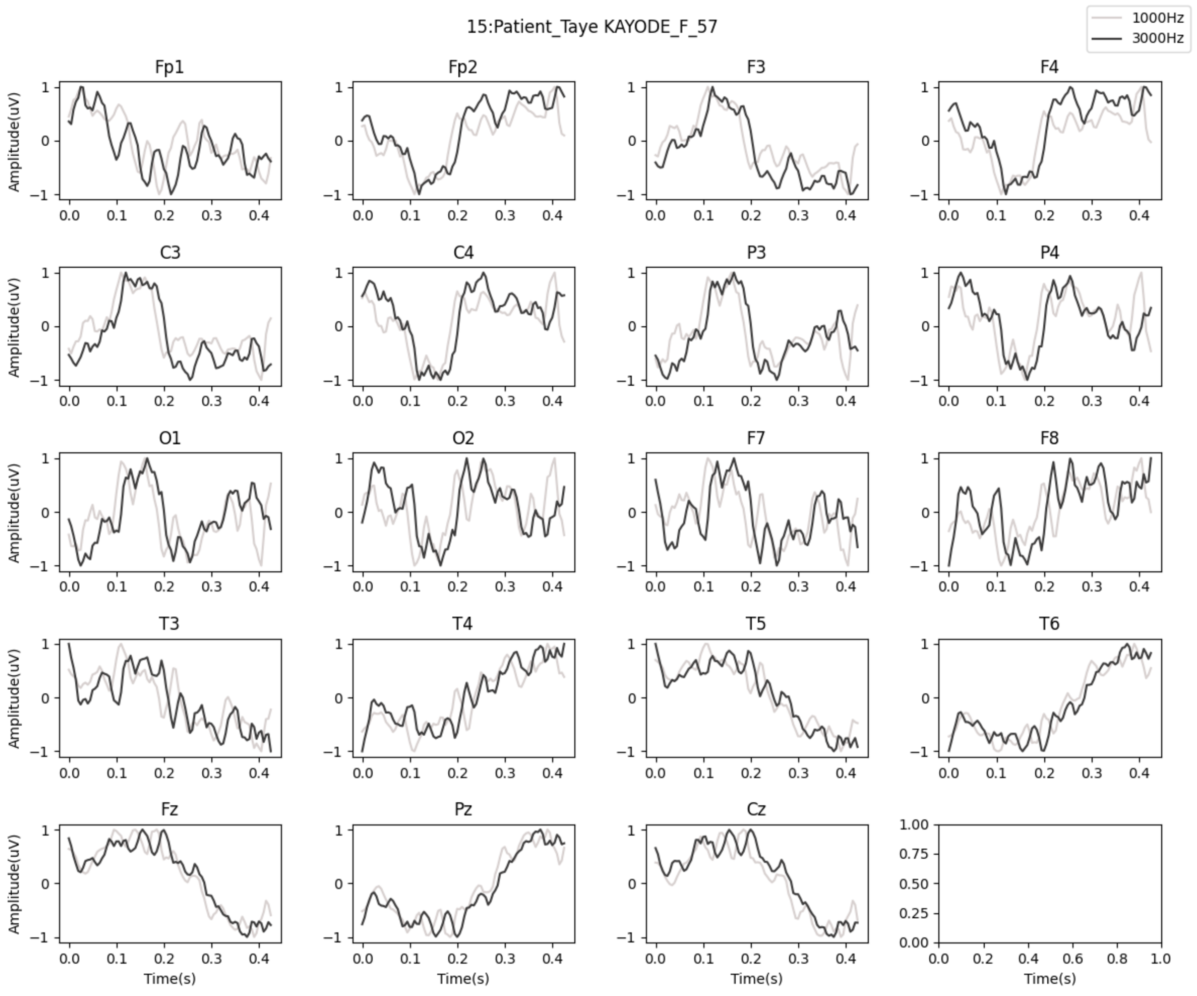

Figure 20: A SZ subject MMN plots

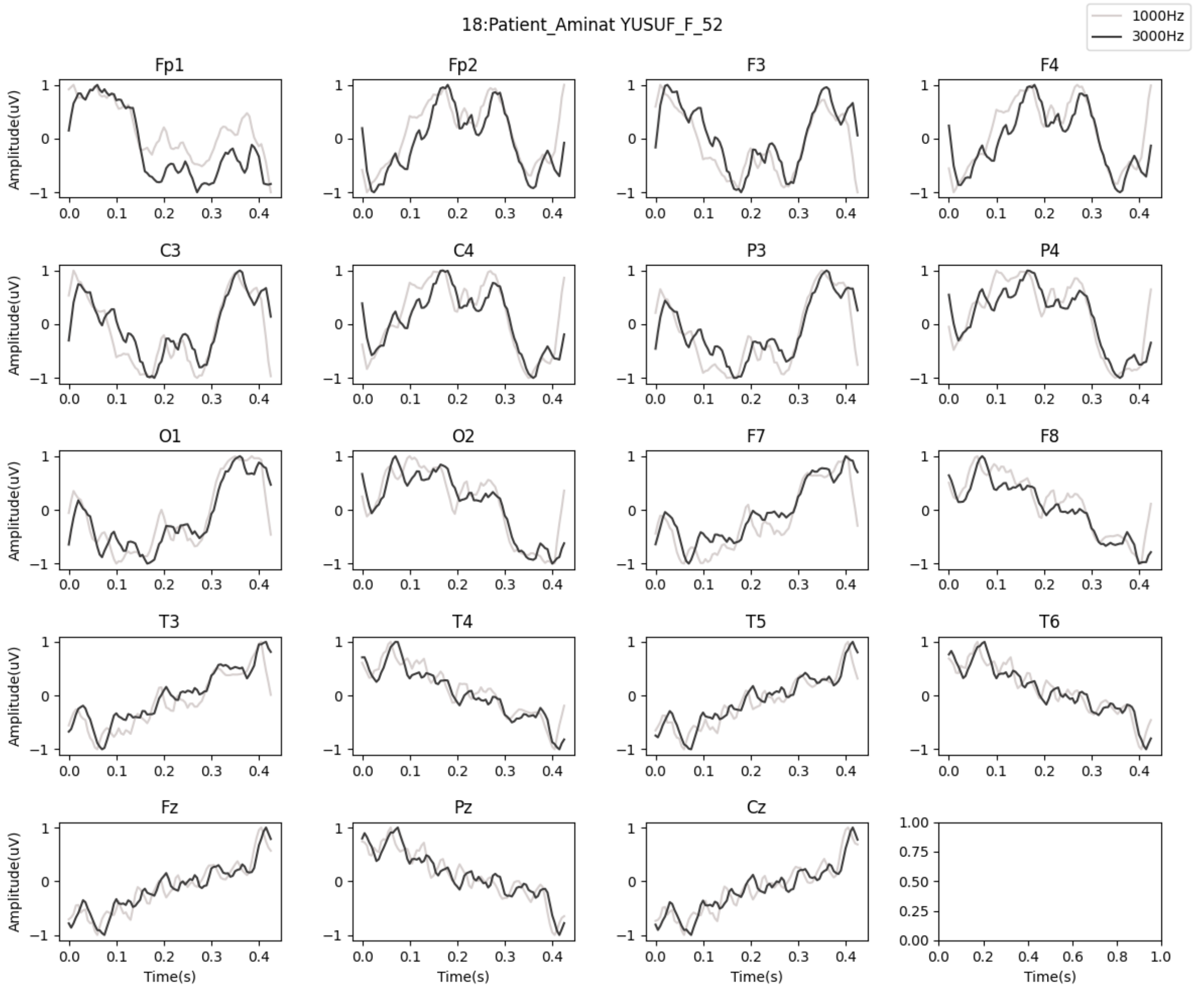

Figure 21: A SZ subject MMN plots

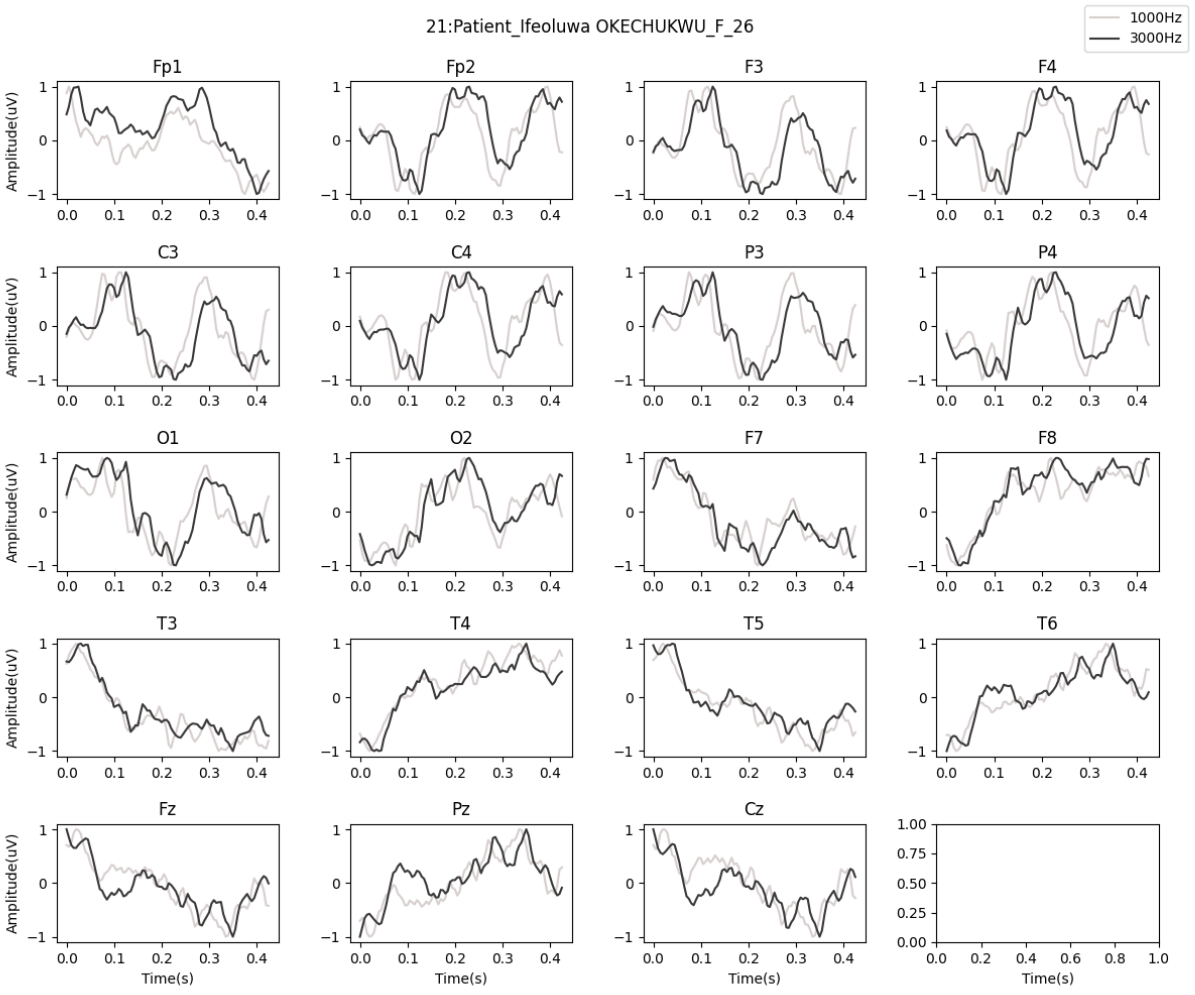

Figure 22: A SZ subject MMN plots

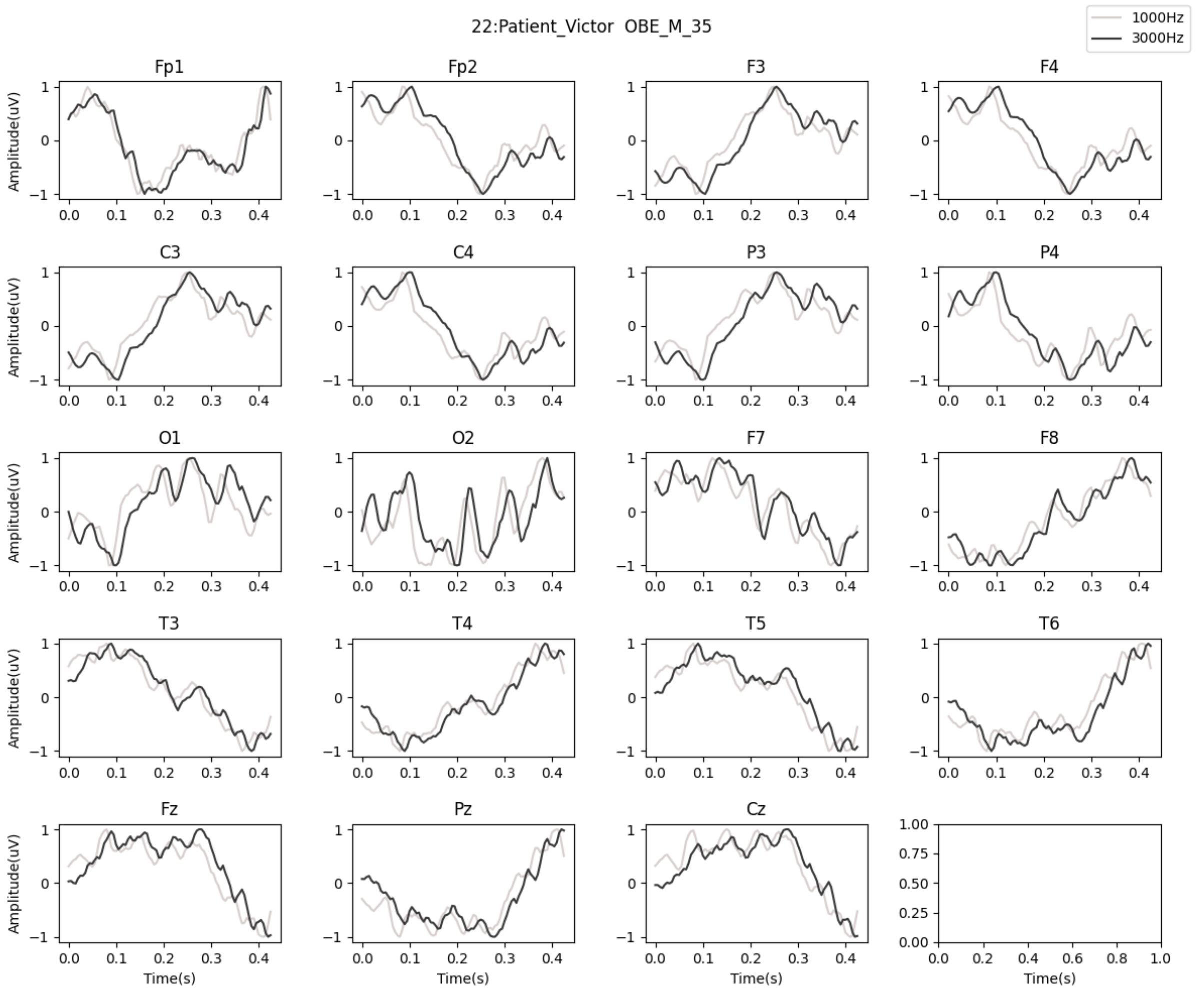

Figure 23: A SZ subject MMN plots

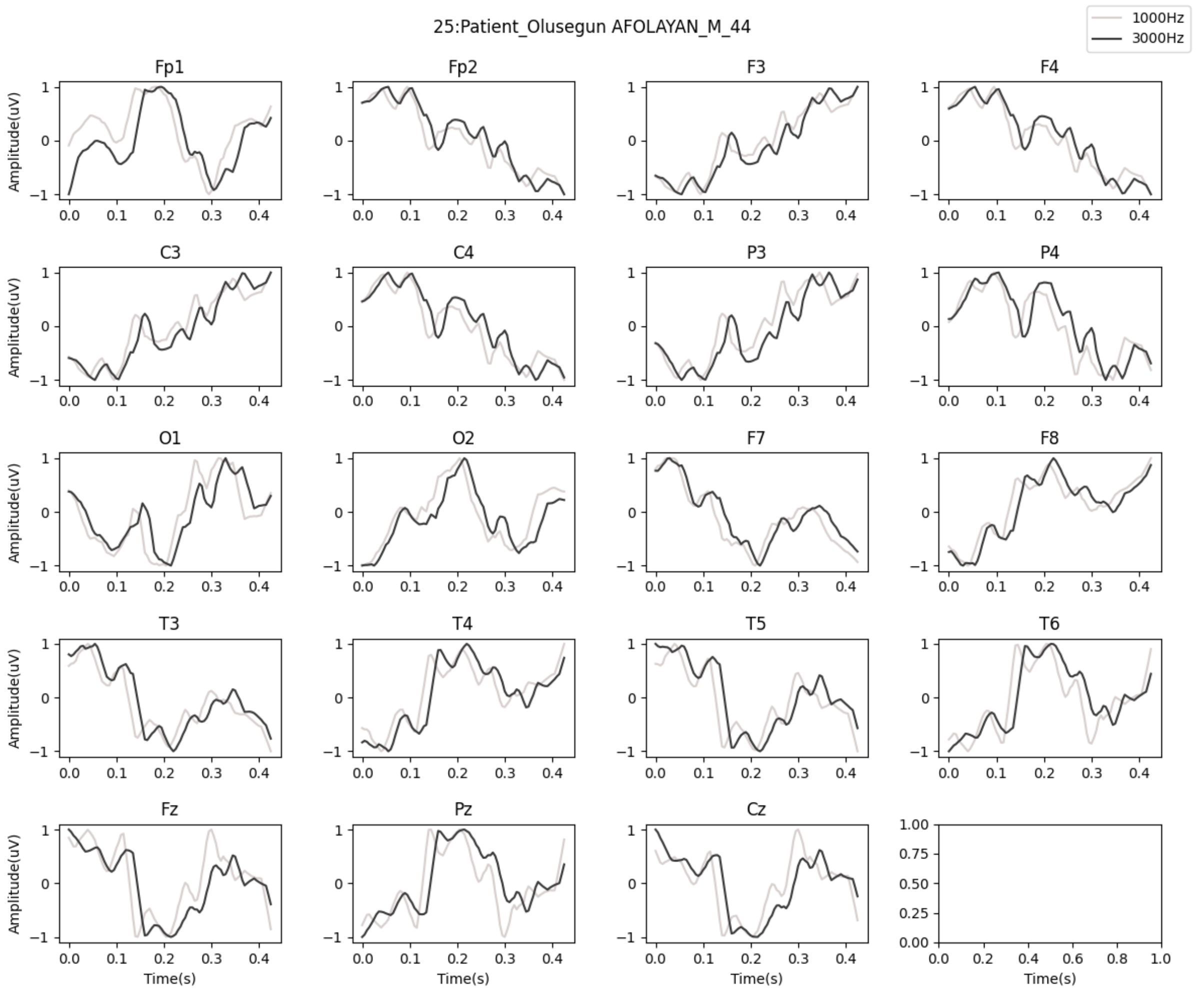

Figure 24: A SZ subject MMN plots

4 Appendix

4.1 Fuzzy Entropy Code

```
import numpy as np
from scipy.spatial.distance import cdist
import time

def sigmoid(x,r):
    assert isinstance(r,tuple), 'When Fx = "Sigmoid", r must be a two-element tuple.'
    y = 1/(1 + np.exp((x-r[1])/r[0]))
    return y
def default(x,r):
    assert isinstance(r,tuple), 'When Fx = "Default", r must be a two-element tuple.'
    y = np.exp(-(x**r[1])/r[0])
    return y
def modsampen(x,r):
    assert isinstance(r,tuple), 'When Fx = "Modsampen", r must be a two-element tuple.
    y = 1/(1 + np.exp((x-r[1])/r[0]))
    return y
def gudermannian(x,r):
    if r <= 0:
        raise Exception('When Fx = "Gudermannian", r must be a scalar > 0.')
    y = np.arctan(np.tanh(r/x))
    y = y/np.max(y)
    return y
def linear(x,r):
    if r == 0 and x.shape[0]>1:
        y = np.exp(-(x - np.min(x))/np.ptp(x))
    elif r == 1:
        y = np.exp(-(x - np.min(x)))
    elif r == 0 and x.shape[0]==1:
        y = 0
    else:
        print(r)
        raise Exception('When Fx = "Linear", r must be 0 or 1')
    return y

class fuzzEntropy:
    def __init__(self,window_size,dissimilarity_index,membership_function='linear'):
        self.m = self.window_size = window_size
        self.r = self.dissimilarity_index = dissimilarity_index
        self.mu = self.membership_function = globals()[membership_function.lower()]

    def __computeFuzzyMatrix(self,x,m):
        if x.ndim == 1:
            N = x.shape[0]
            Xm = np.array([x[i:i+m-1].tolist() for i in range(0,N-m)])
            dm = cdist(Xm,Xm,'euclidean')
            dm = self.mu(dm,self.r)
            phim = np.sum(dm,axis=1)/(N-m+1)
        return phim

    def _fuzzyEntropyCompute(self,x):
        N = x.shape[0]
        phim = self.__computeFuzzyMatrix(x,self.m)
        phim1 = self.__computeFuzzyMatrix(x,self.m+1)

        psim = (1/(N-self.m+1)) * np.sum(phim,axis=0)
        psim1 = (1/(N-self.m+2)) * np.sum(phim1,axis=0)

        with np.errstate(divide='ignore', invalid='ignore'):
            fuzz = np.log(psim)-np.log(psim1)
        return fuzz

def fuzzEntropy2D(x,window_size,dissimilarity_index,membership_function=gaussianMembe
        fuzzyent = fuzzEntropy(window_size,dissimilarity_index,membership_function)

        res = np.empty(x.shape[0])
        for i in range(x.shape[0]):
            res[i] = fuzzyent._fuzzyEntropyCompute(x[i,:])
        return res.mean()
```