ESCUELA UNIVERSITARIA POLITÉCNICA
DE LA ALMUNIA DE DOÑA GODINA (ZARAGOZA)

ANEXOS

Estudio de Viabilidad del Desarrollo de una BCI Aplicada a un Sistema Informático

BCI Development Feasibility Research Applied to a Computer System

424.20.48

Autor: Iñigo Garaboa Cotelo

Director: Dr. David Asiain Ansorena

Fecha: 25 de noviembre de 2020

INDICE DE CONTENIDO

INDICE DE TABLAS

INDICE DE ESQUEMAS DE LABVIEW

1. CÓDIGOS DE PROGRAMACIÓN

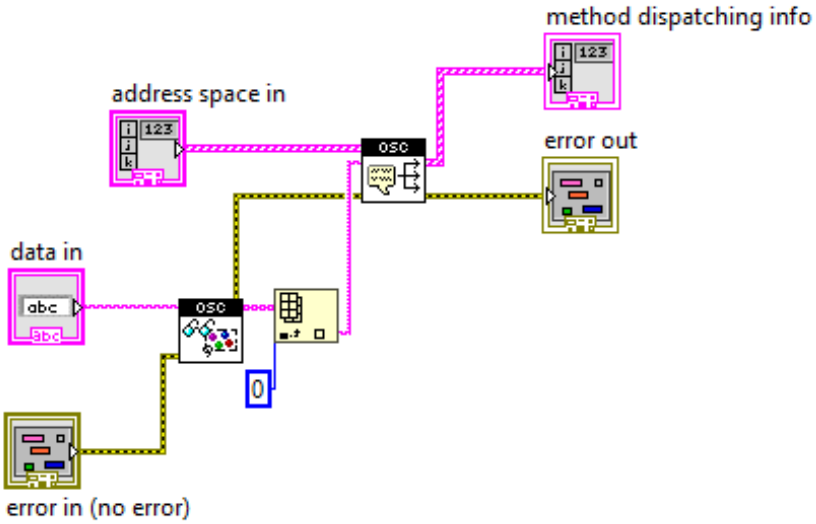
1.1. CÓDIGOS DE LABVIEW

1.1.1. Funciones de Diseño Propio

1.1.1.1. Función "Bundle to Messaje Dispatched"

Esquema	Descripción
 ***Esquema 1: Bloque de la función "Bundle to Message Dispatched"***	Transforma los datos extraídos del puerto UDP en bruto, obteniendo como resultado los mensajes identificados con sus respectivas direcciones.

Programa

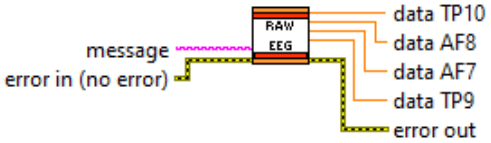


Esquema 2: Esquema interno de la función "Bundle to Message Dispatched"

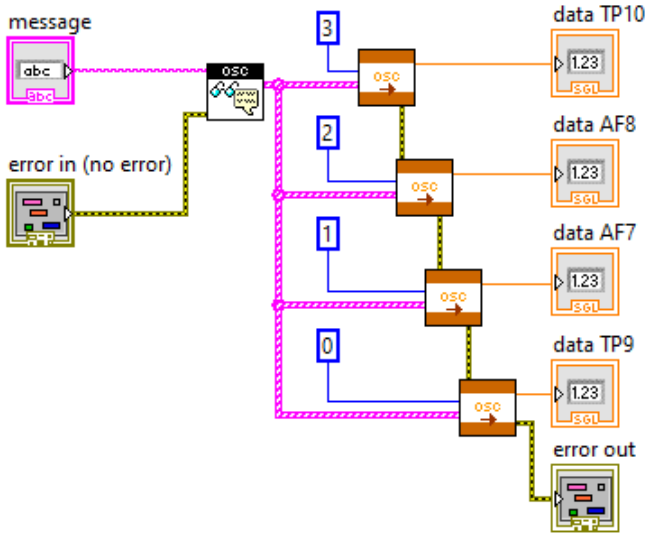
Entradas	Información	Salidas	Información
Address Space In	Array con las direcciones a leer habiéndose aplicado la función "Create Space Address"	Method Dispatching Info	*Array* con los mensajes y direcciones que sean iguales a las solicitadas previamente
Data In	*String* que procede del puerto UDP		
Error In	Errores antes de que se ejecute la función	Error Out	Errores una vez sea ejecutada la función

Tabla 1: Información de la función "Bundle to Message Dispatched"

1.1.1.2. Función "RAW EEG Adquisition"

Esquema	Descripción
 ***Esquema 3: Bloque de la función "RAW EEG Adquisition"***	Adquiere los datos de las señales EEG en bruto captadas por los cuatro canales de registro del Muse.

Programa

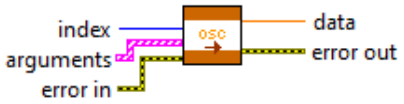


Esquema 4: Esquema interno de la función "RAW EEG Adquisition"

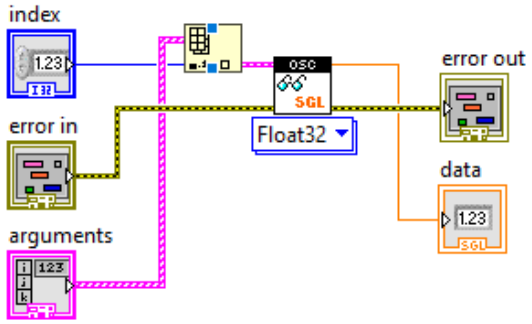
Entradas	Información	Salidas	Información
Message	Mensaje a decodificar	Data TP10	Valor de la tensión en bruto procedente del canal TP10 en coma flotante y en µV
Error In	Errores antes de que se ejecute la función	Data AF8	Valor de la tensión en bruto procedente del canal AF8 en coma flotante y en µV
		Data AF7	Valor de la tensión en bruto procedente del canal AF7 en coma flotante y en µV
		Data TP9	Valor de la tensión en bruto procedente del canal TP9 en coma flotante y en µV
		Error Out	Errores una vez sea ejecutada la función

Tabla 2: Información de la función "RAW EEG Adquisition"

1.1.1.3. Función "OSC to Float"

Esquema	Descripción
 ***Esquema 5: Bloque de la función "OSC to Float"***	Convierte el argumento que posee el valor de tensión de una señal en bruto de un canal en su equivalente en coma flotante.

Programa



Esquema 6: Esquema interno de la función "OSC to Float"

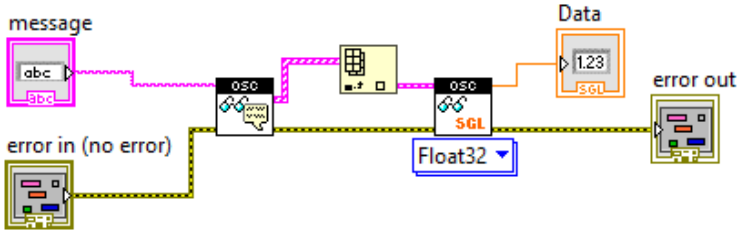
Entradas	Información	Salidas	Información
Index	Posición del valor a procesar del *array* "*Arguments*".	Data	Valor de la tensión en bruto.
Arguments	*Array* de *strings* compuesto por los argumentos del mensaje una vez decodificado.	Error Out	Errores una vez ejecutada la función
Error In	Errores antes de ejecutar función		

Tabla 3: Información de la función "OSC to Float"

1.1.1.4. Función "Horseshoe"

Esquema	Descripción
 ***Esquema 7: Bloque de la función "Horseshoe"***	Adquiere información acerca de la comunicación entre la persona que emplea el dispositivo Muse y los canales de registro.

Programa

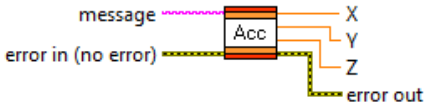
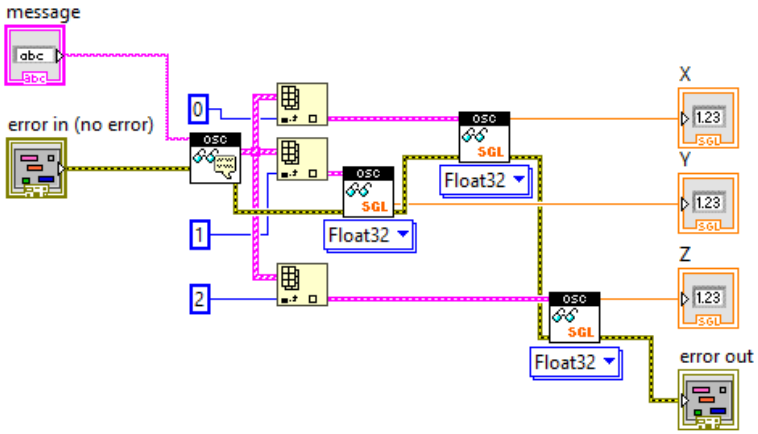


Esquema 8: Esquema interno de la función "Horseshoe"

Entradas	Información	Salidas	Información
Message	Mensaje a decodificar.	Data	Información de la calidad de la comunicación con los canales. - 1 = Buena - 2 = Media - 4 = Mala
Error In	Errores antes de que se ejecute la función		
		Error Out	Errores una vez ejecutada la función

Tabla 4: Información de la función "Horseshoe"

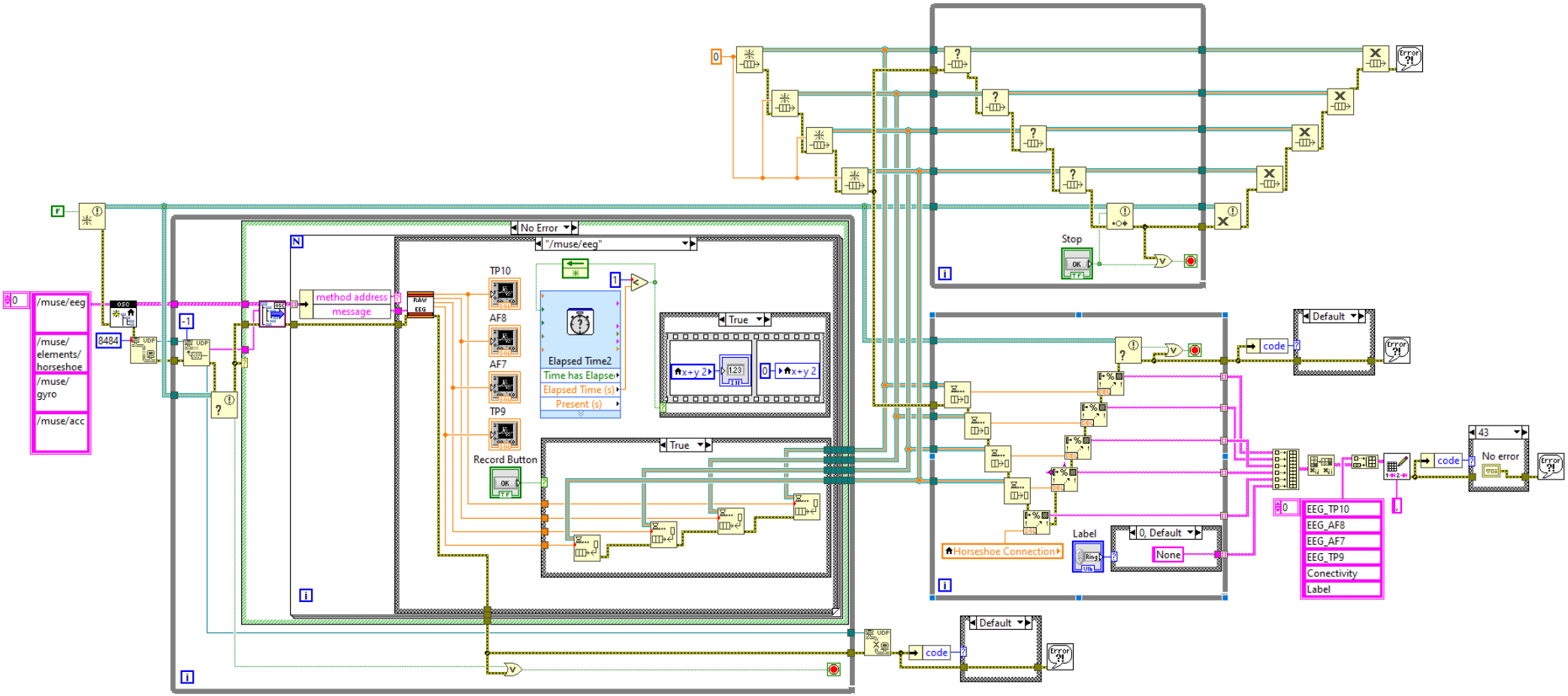
1.1.1.5. Función "Gyroscope" y "Accelerometer"

Esquemas	Descripción
 ***Esquema 9: Bloque de la función "Gyroscope"***	Adquiere los datos captados por el giroscopio y el acelerómetro que incluye Muse 2.
 ***Esquema 10: Bloque de la función "Accelerometer"***	
Programa	
 ***Esquema 11 Esquema interno de las funciones "Gyroscope" y "Accelerometer"***	

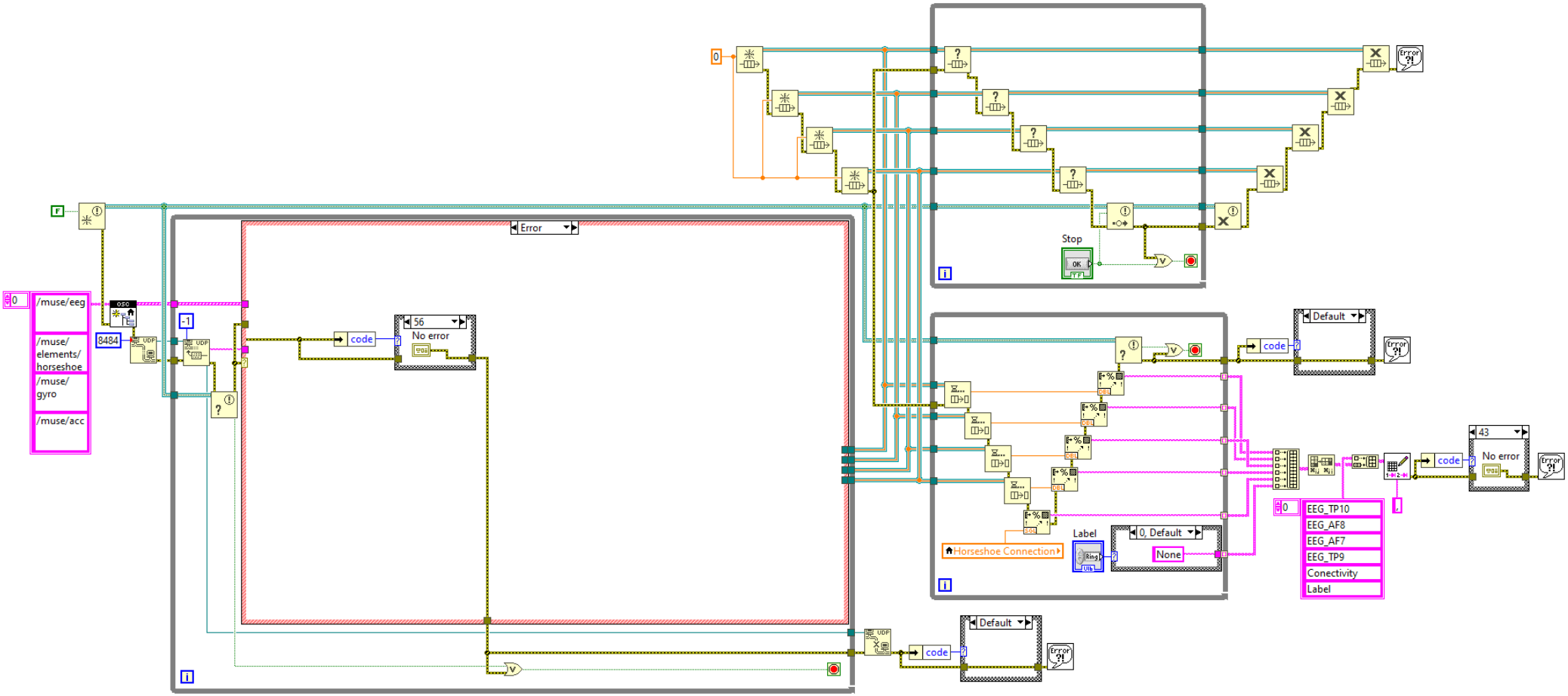
Entradas	Información	Salidas	Información
Message	Mensaje a decodificar.	X	Valores en el eje de X.
Error In	Errores antes de que se ejecute la función	Y	Valores en el eje de Y.
		Z	Valores en el eje de Z.
		Error Out	Errores al ejecutar la función

Tabla 5: Información de las funciones "Gyroscope" y "Accelerometer"

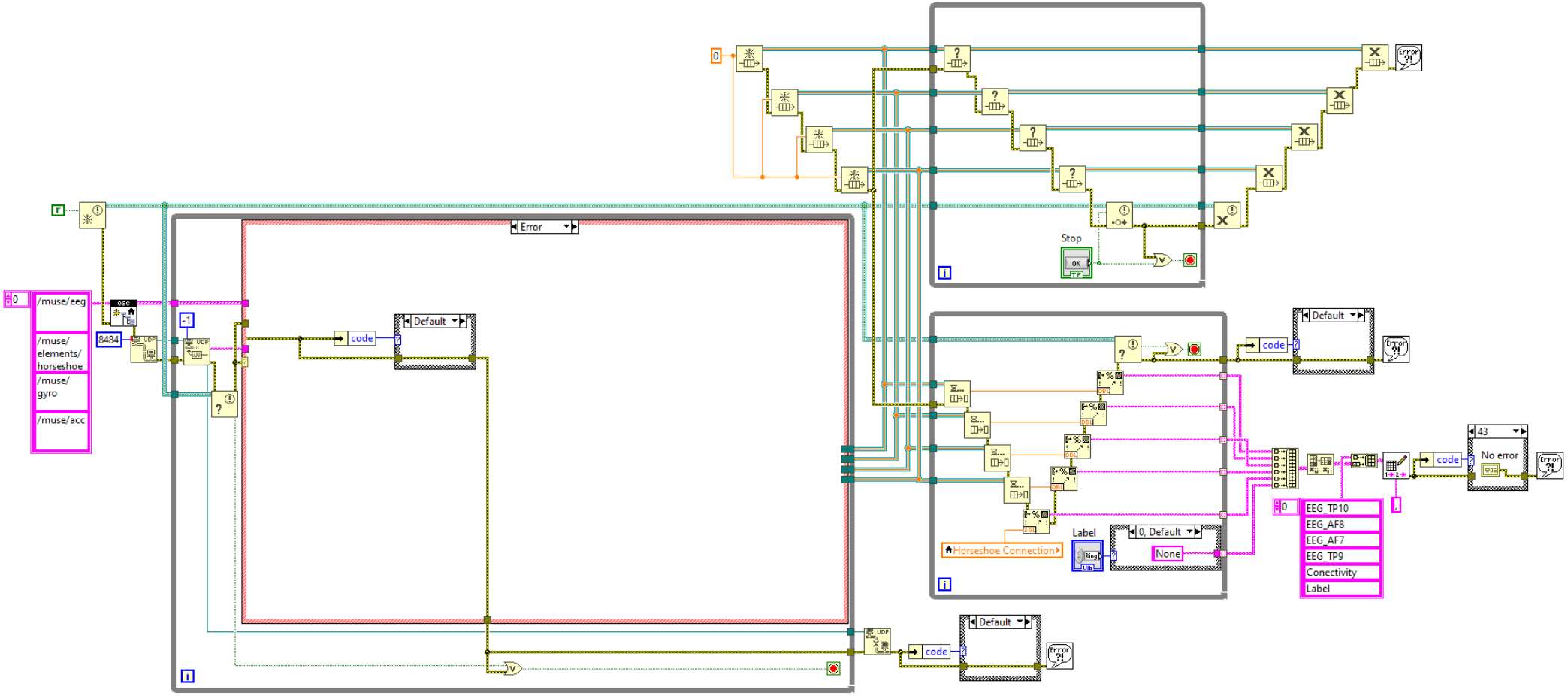
1.1.2. Programa General



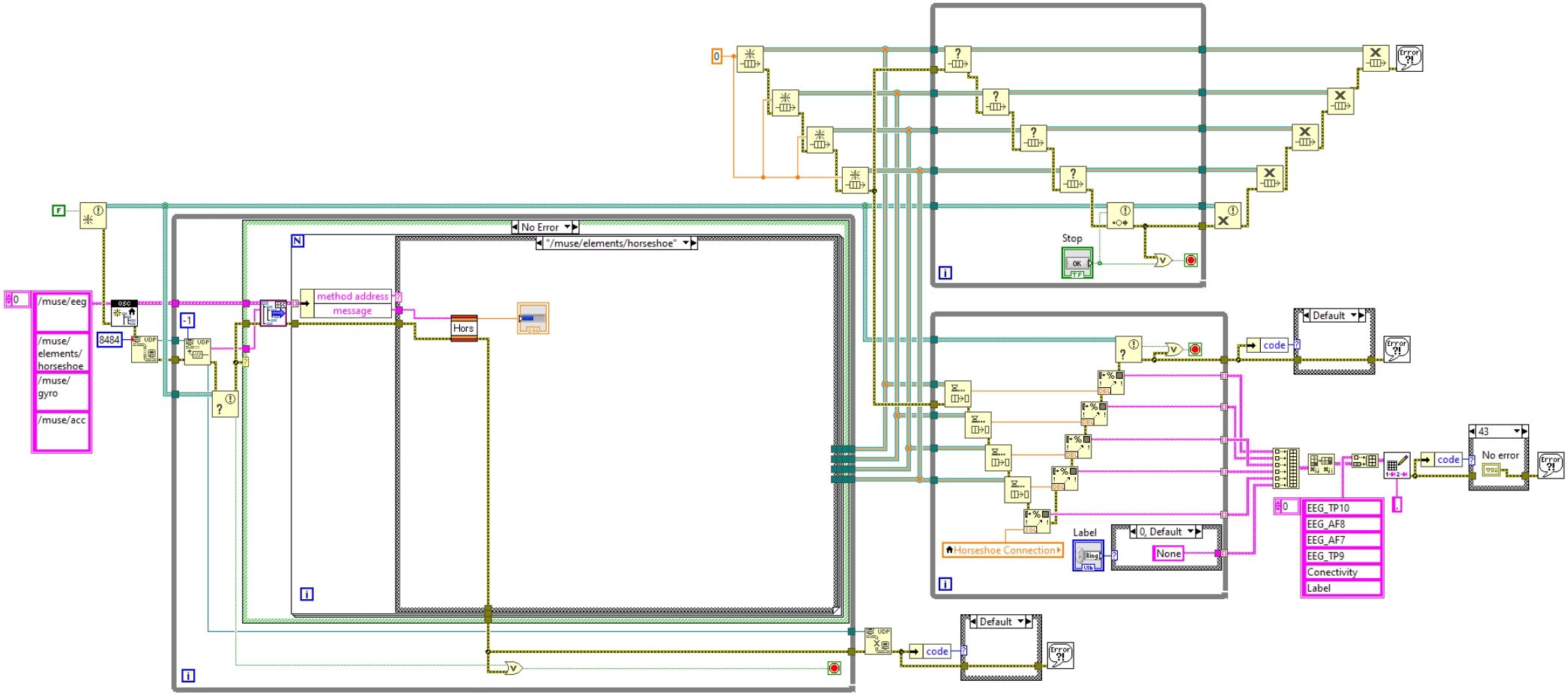
Esquema 12: Programa general con el método para leer las señales EEG



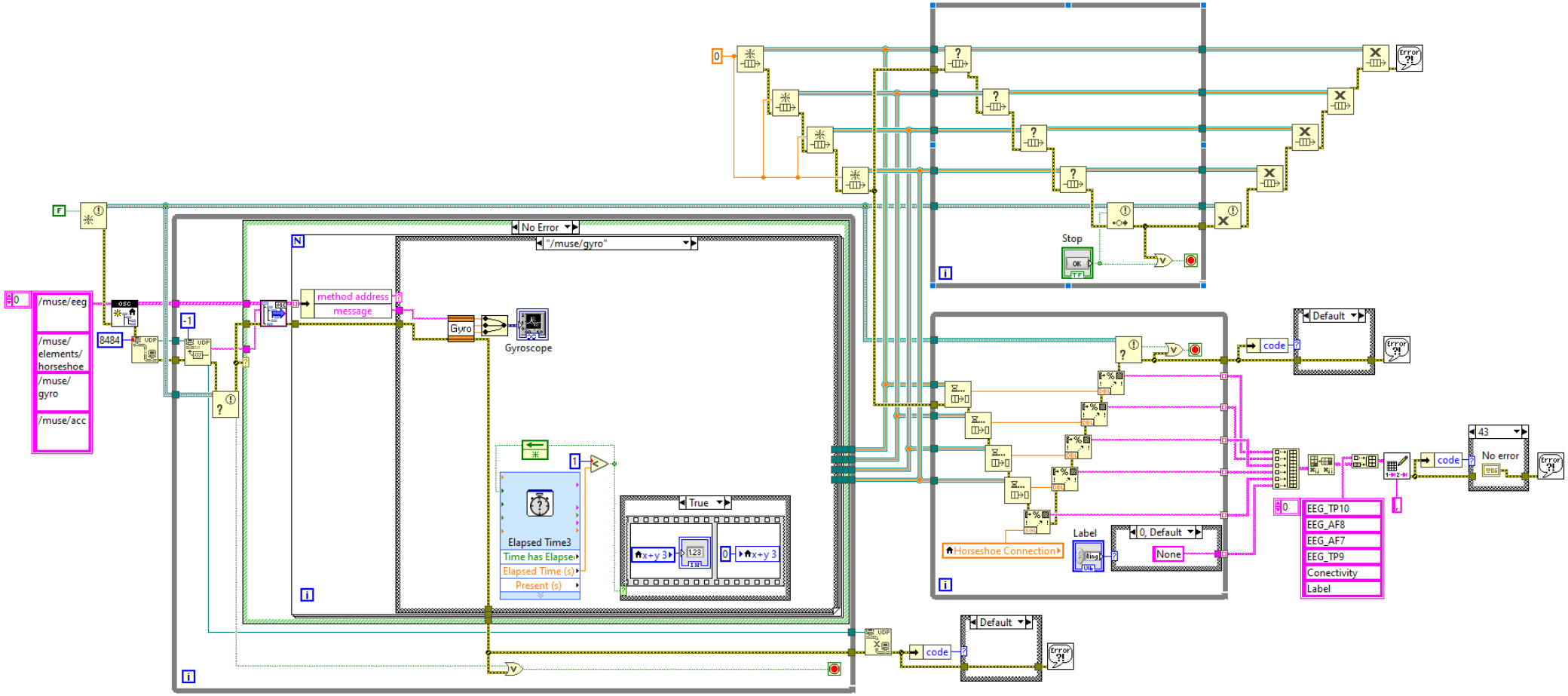
Esquema 13: Programa general cuando se produce un error 56 en el bloque "UDP Read"



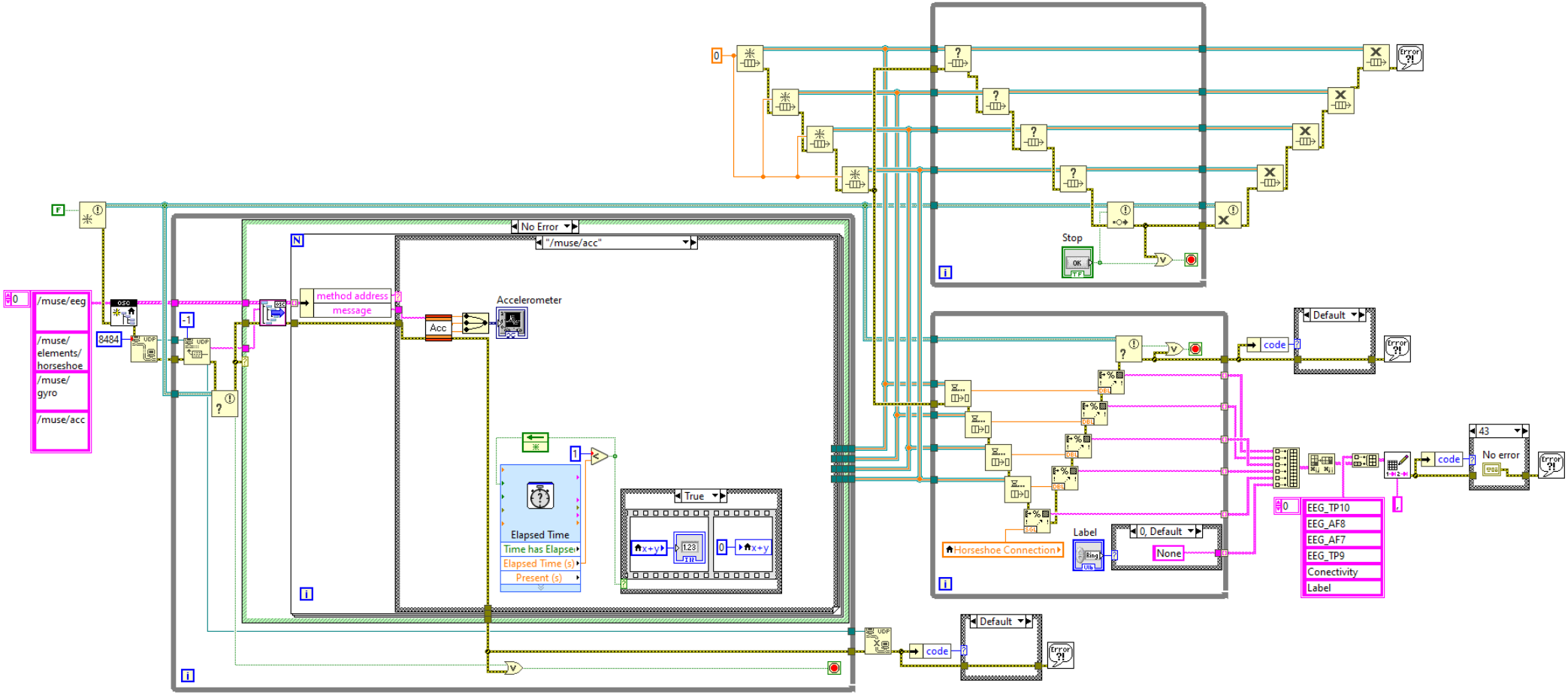
Esquema 14: Programa general cuando se produce un error en el bloque "UDP Read" pero no es 56



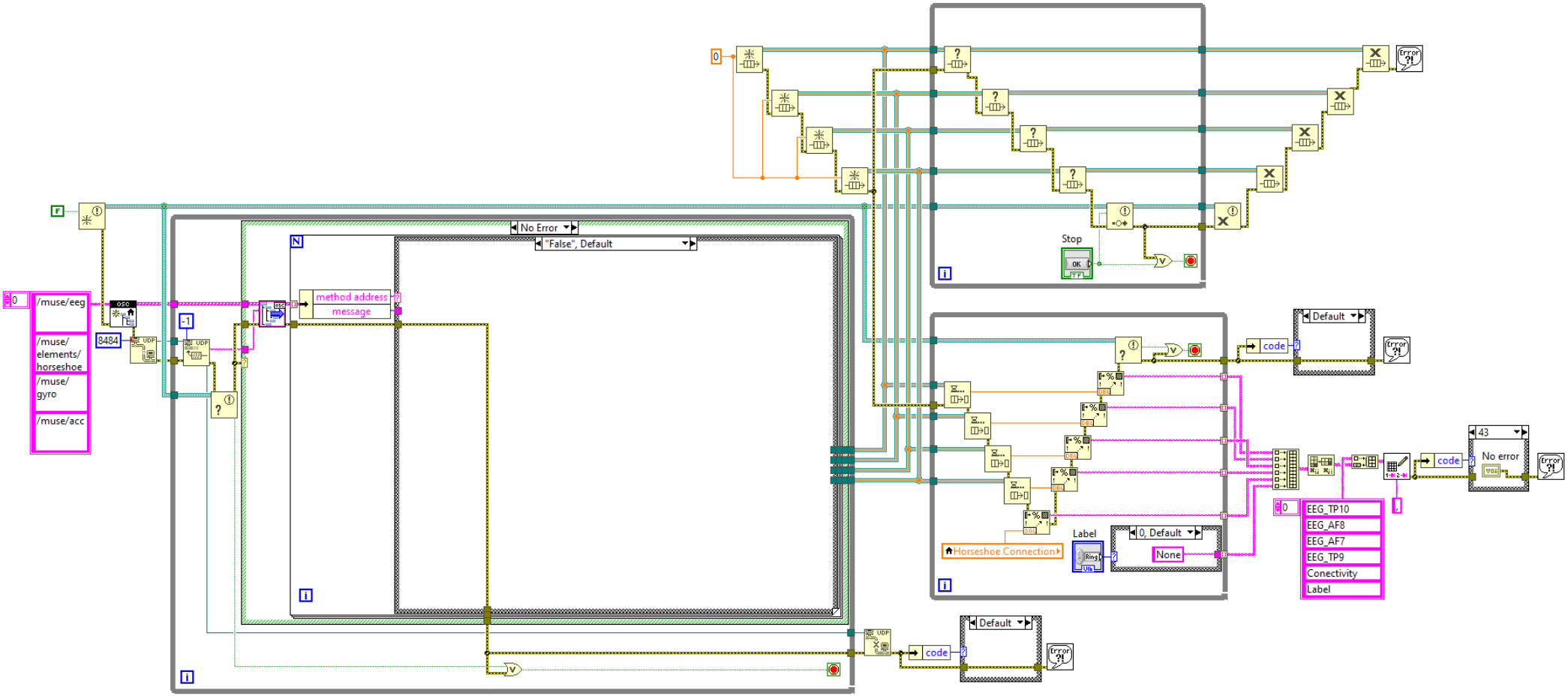
Esquema 15: Programa general con el método para leer la calidad de contacto de los sensores



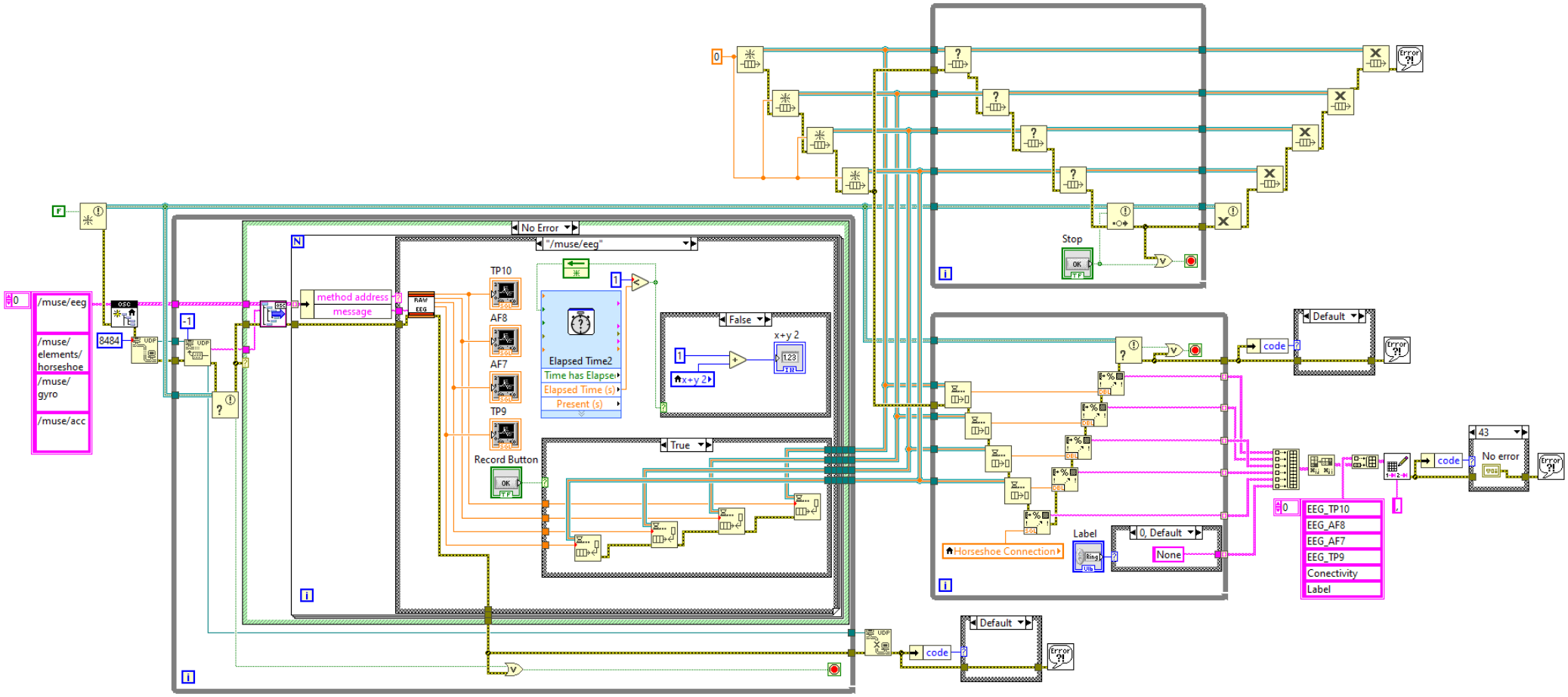
Esquema 16: Programa general con el método para leer la información del giroscopio



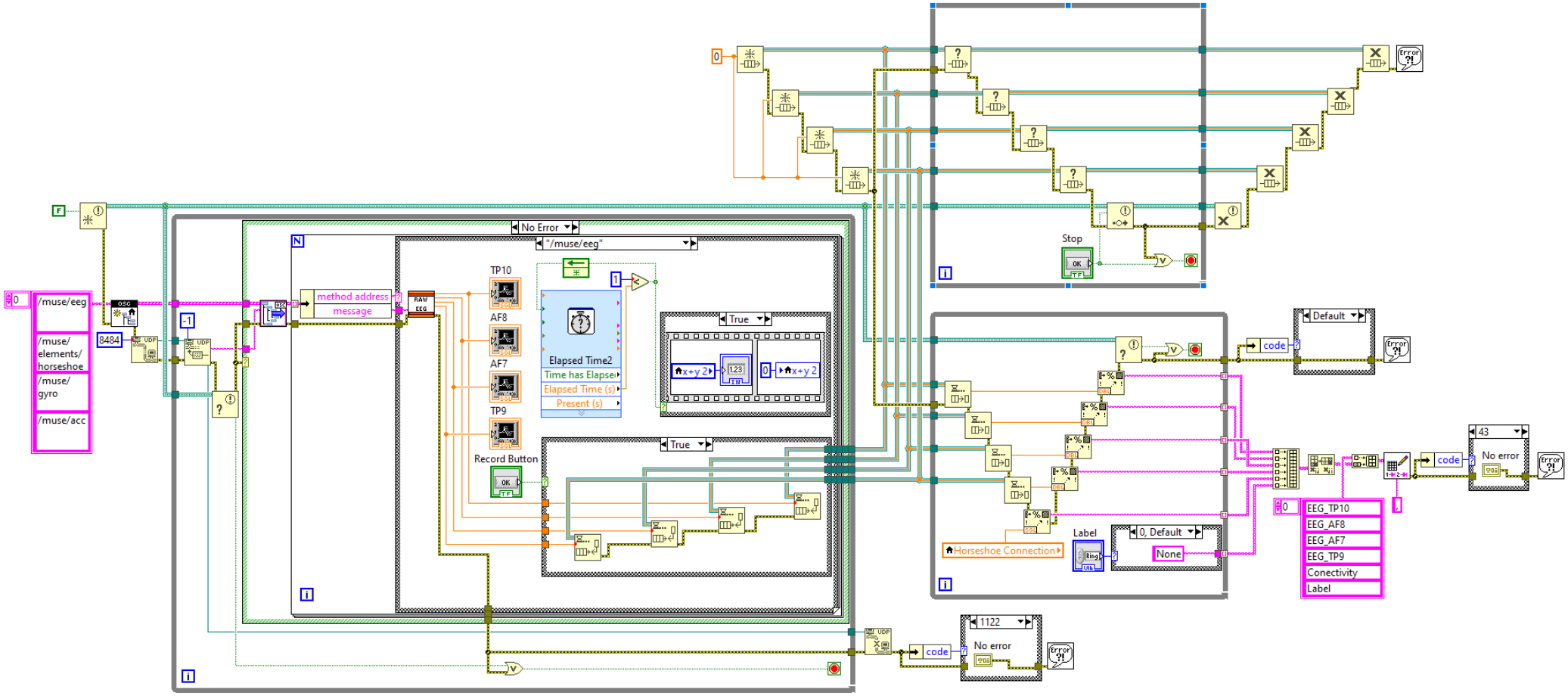
Esquema 17: Programa general con el método para leer la información del acelerómetro



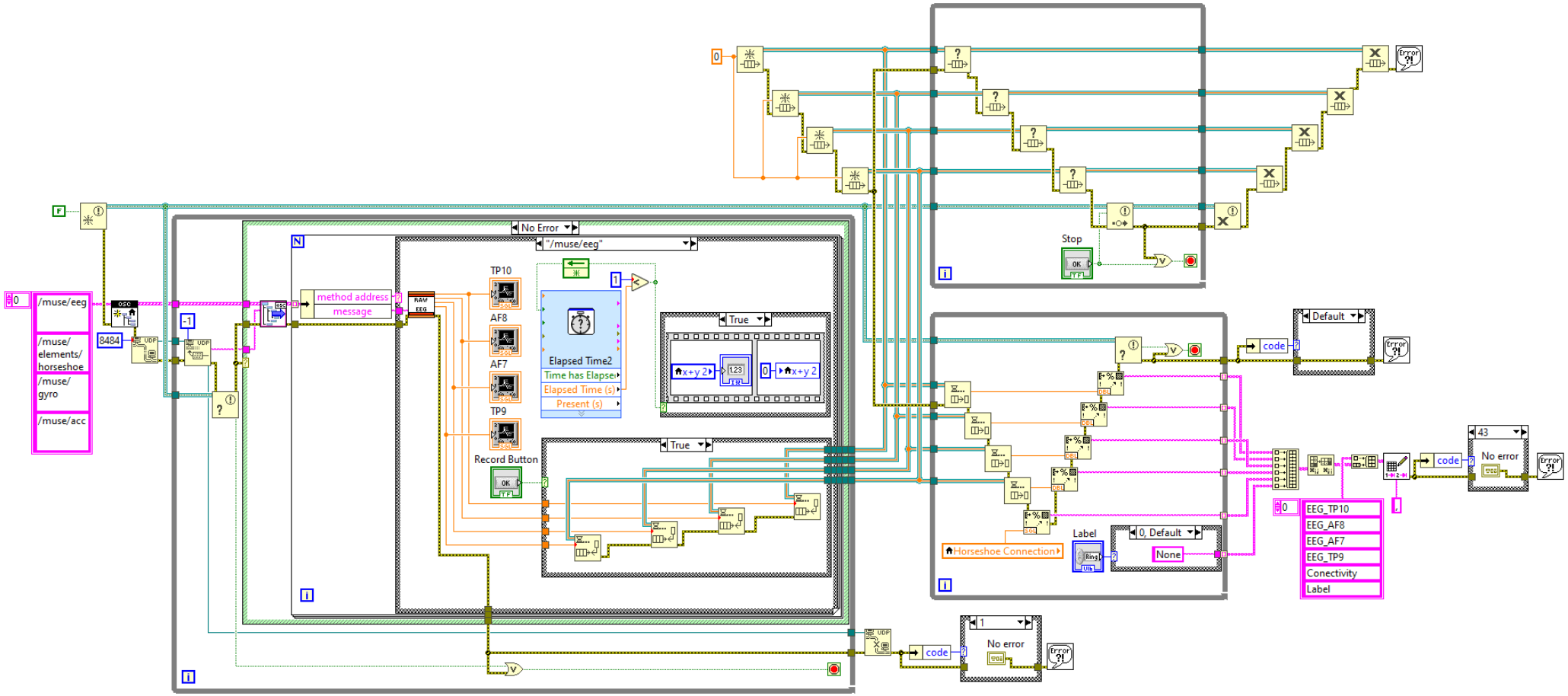
Esquema 18: Programa general si la dirección OSC no posee ningún método diseñado



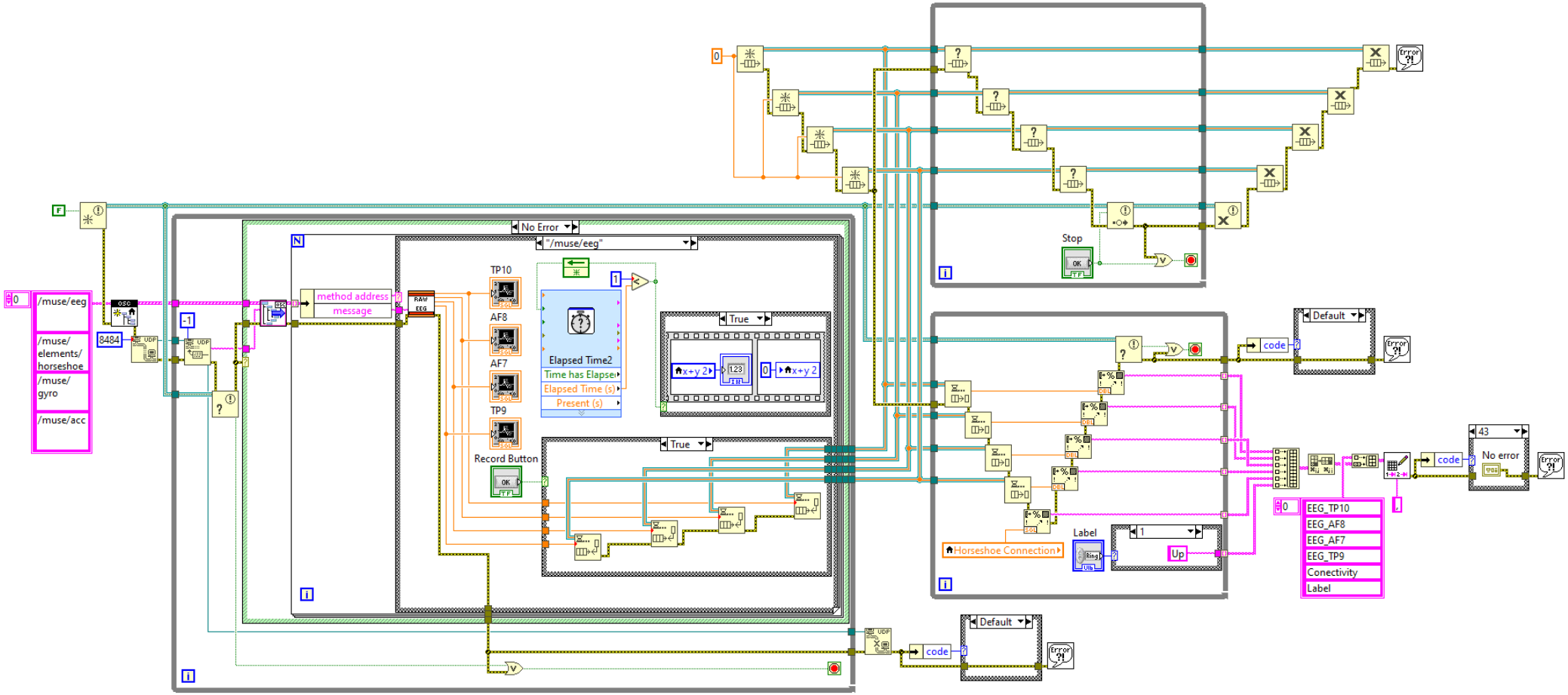
Esquema 19: Programa general con el método para leer las señales EEG al pasar un segundo (conteo de las muestras por segundo)



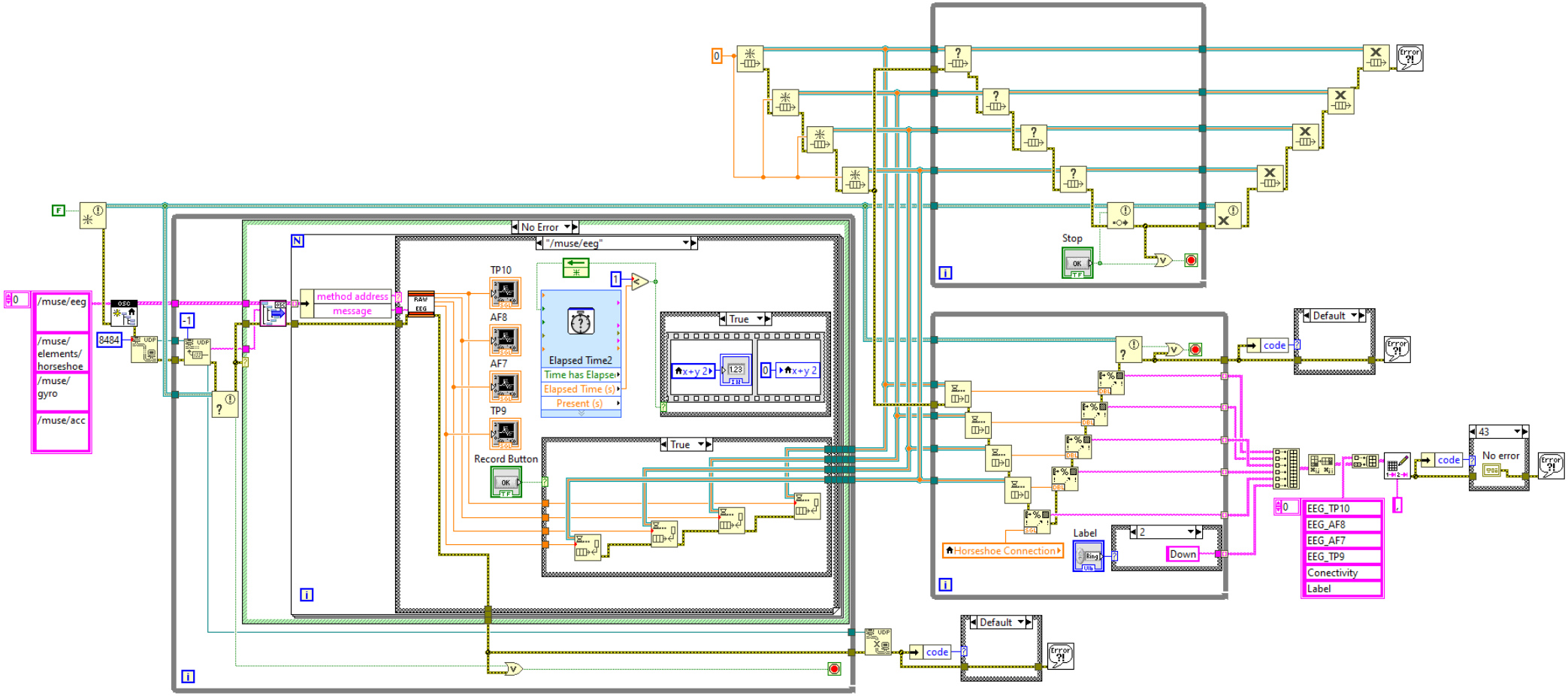
Esquema 20: Programa general si se ha producido un error 1122 dentro del primer bucle "while"



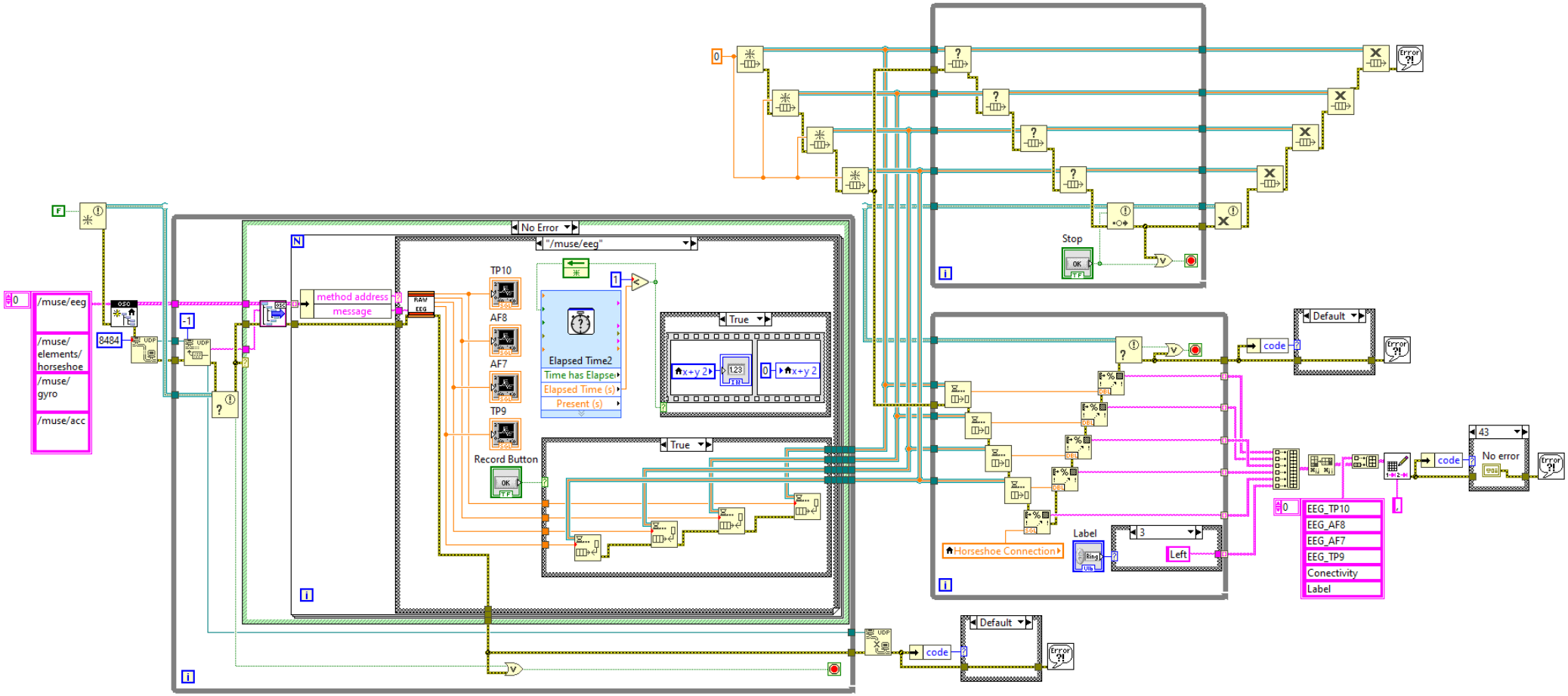
Esquema 21: Programa general si se ha producido un error 1 dentro del primer bucle "while"



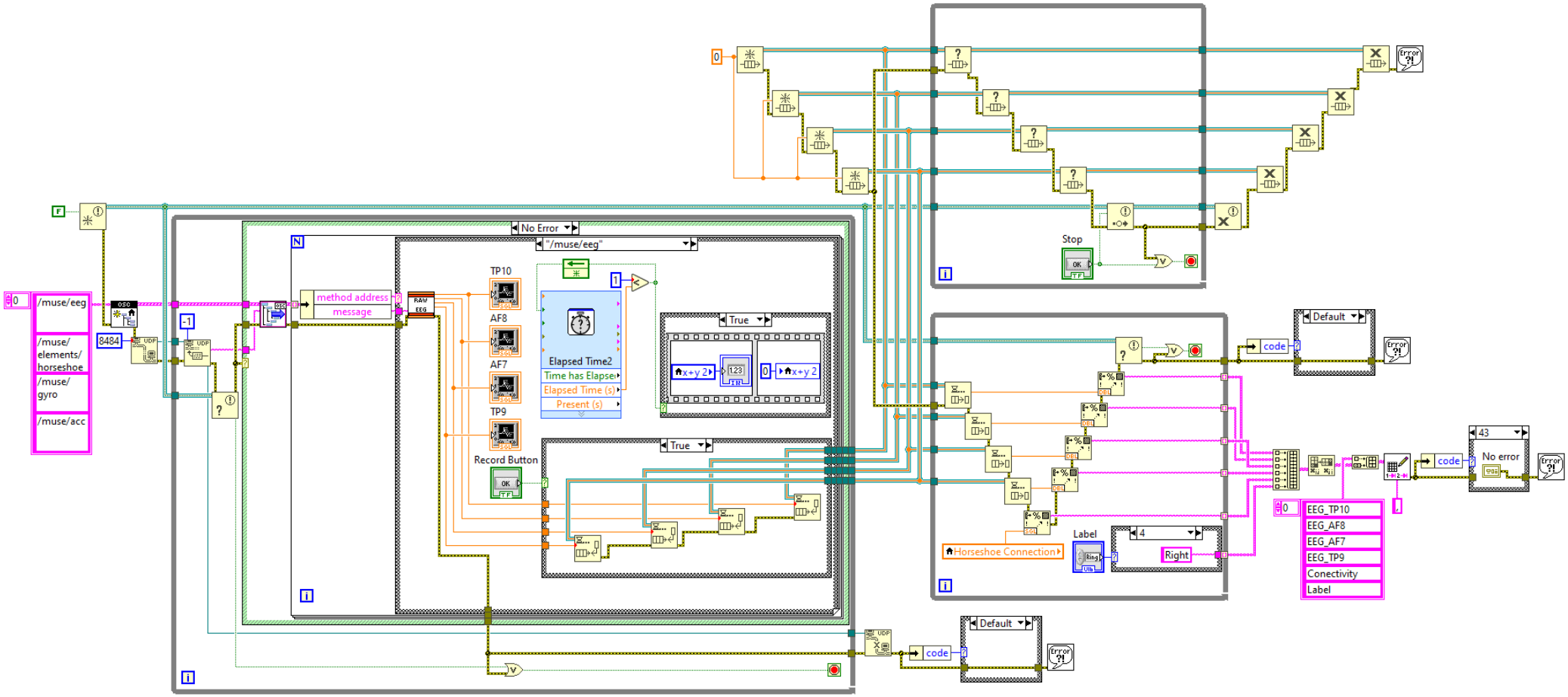
Esquema 22: Programa general cuando se registra la etiqueta "Up"



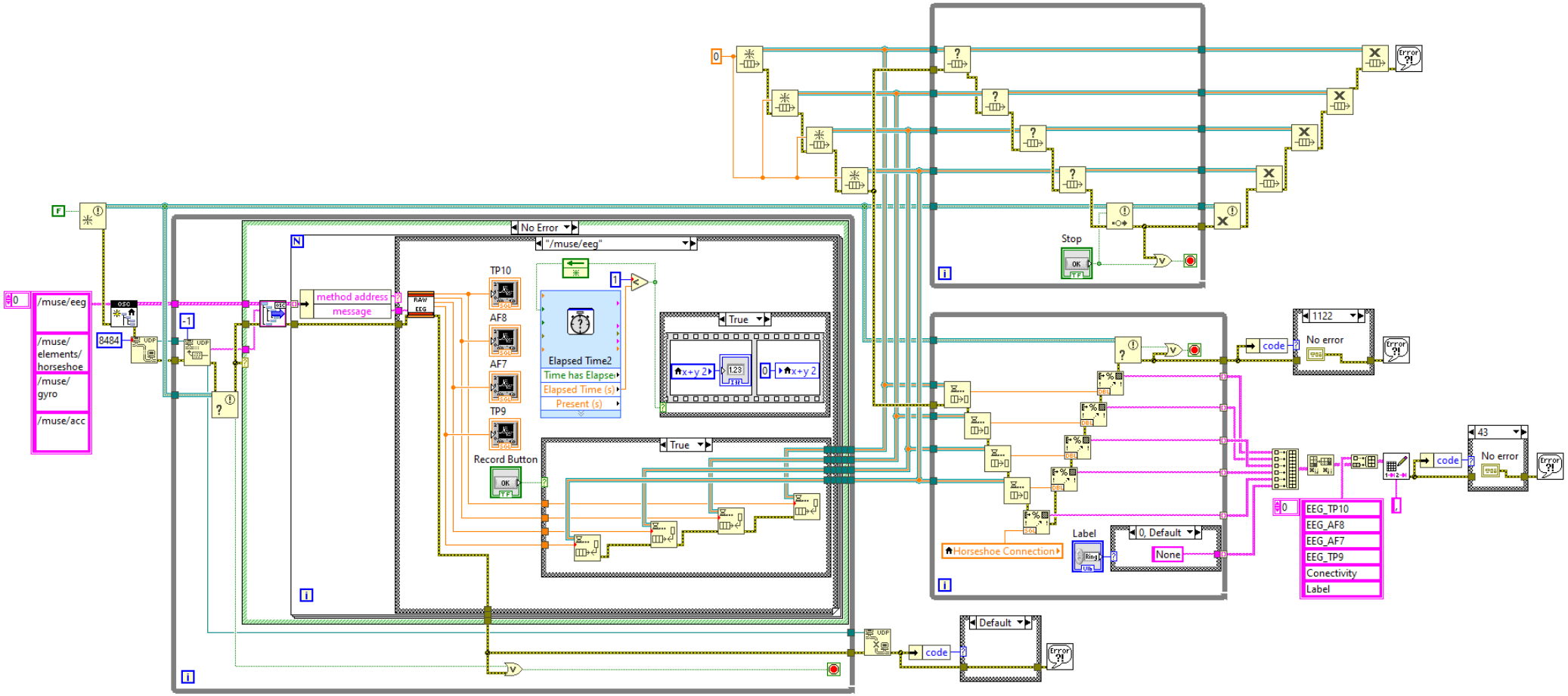
Esquema 23: Programa general cuando se registra la etiqueta "Down"



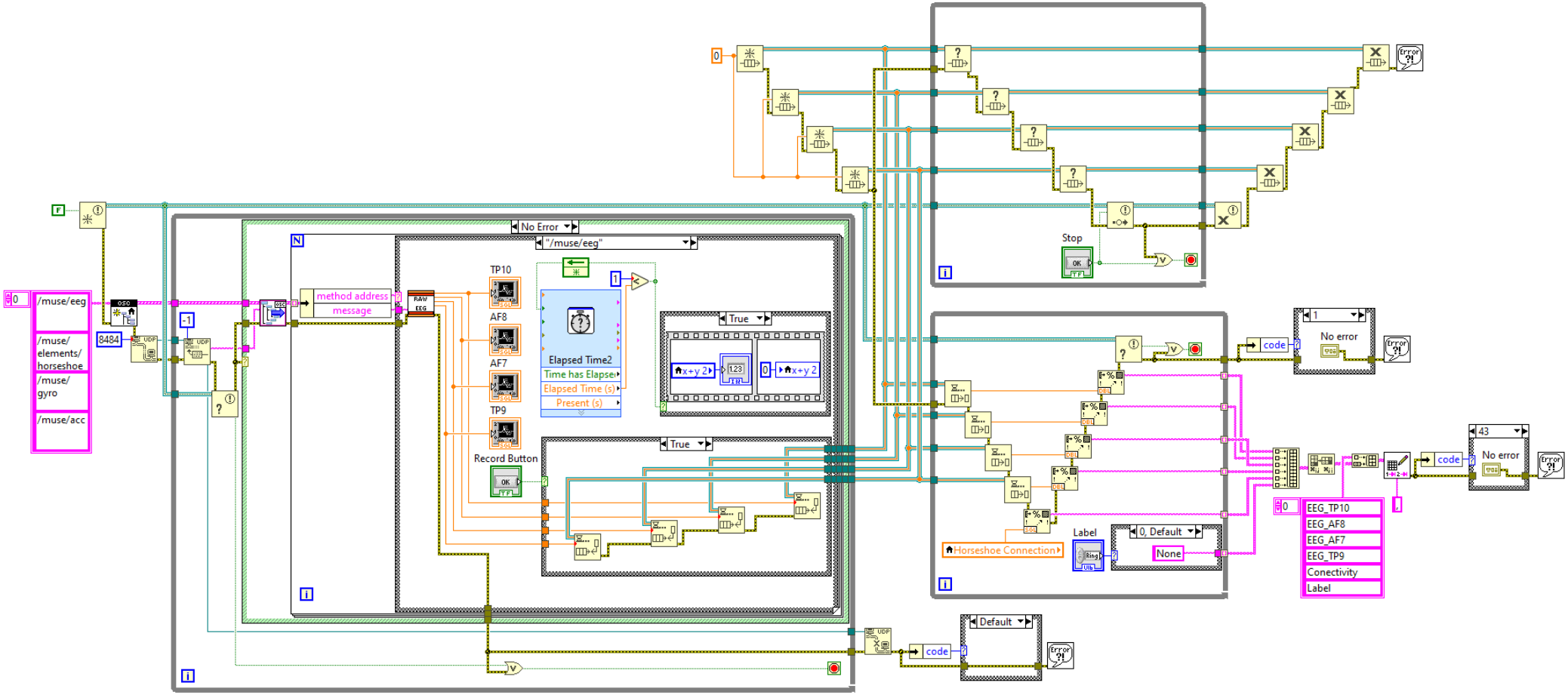
Esquema 24: Programa general cuando se registra la etiqueta "Left"



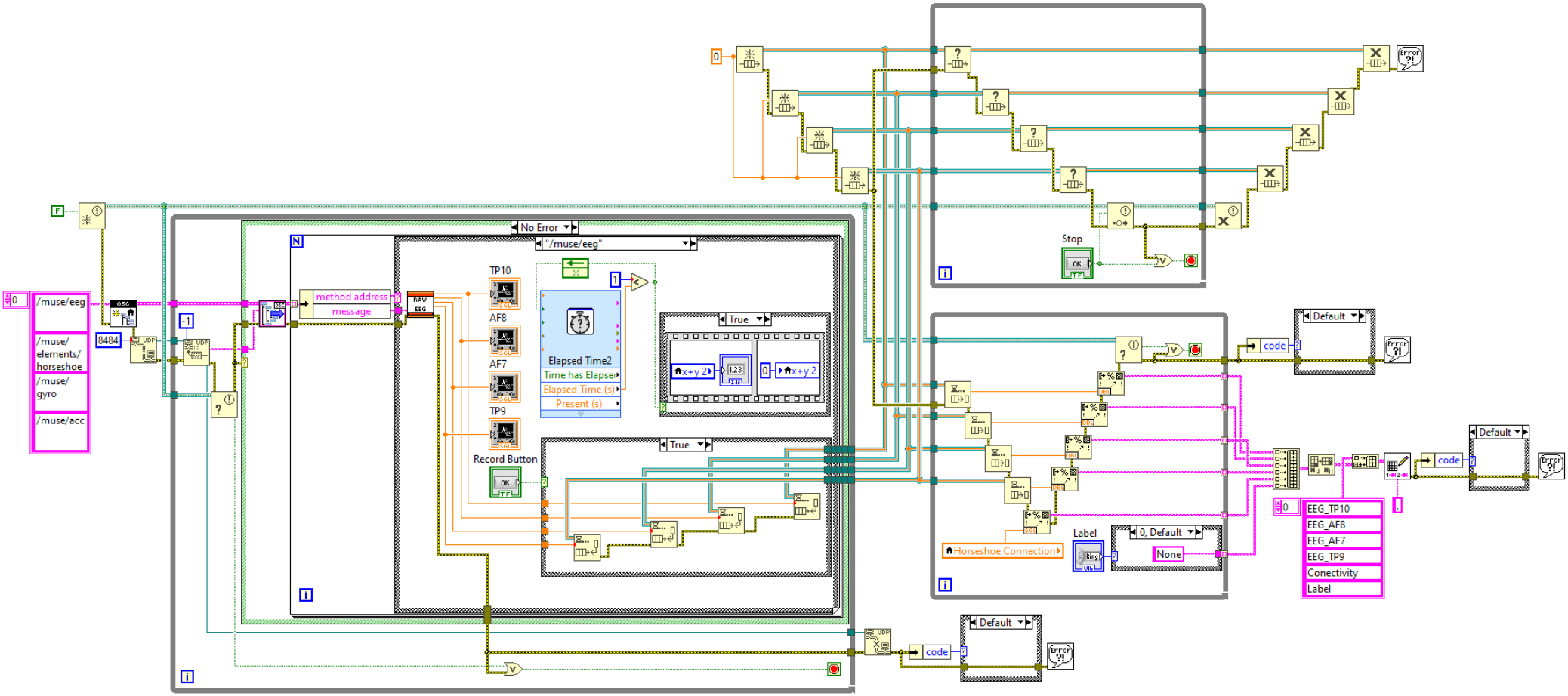
Esquema 25: Programa general cuando se registra la etiqueta "Right"



Esquema 26: Programa general si se ha producido un error 1122 dentro del segundo bucle "while"



Esquema 27: Programa general si se ha producido un error 1 dentro del segundo bucle "while"



Esquema 28: Programa general si se ha producido un error al generar el database si éste no es el 43

1.2. CÓDIGOS DE PYTHON

1.2.1. Agrupamiento de Databases

```
#IMPLEMENTACIÓN DE LIBRERÍAS
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import json
import pandas as pd

#SUEJETO 1
#PRUEBA 1
s111 = pd.read_csv("s1_1_1.csv")
s112 = pd.read_csv("s1_1_2.csv")
s113 = pd.read_csv("s1_1_3.csv")
s114 = pd.read_csv("s1_1_4.csv")
s115 = pd.read_csv("s1_1_5.csv")
s116 = pd.read_csv("s1_1_6.csv")
s117 = pd.read_csv("s1_1_7.csv")
s118 = pd.read_csv("s1_1_8.csv")
s119 = pd.read_csv("s1_1_9.csv")
s1110 = pd.read_csv("s1_1_10.csv")
s1111 = pd.read_csv("s1_1_11.csv")
s1112 = pd.read_csv("s1_1_12.csv")
s1113 = pd.read_csv("s1_1_13.csv")
s1114 = pd.read_csv("s1_1_14.csv")
s1115 = pd.read_csv("s1_1_15.csv")
s1116 = pd.read_csv("s1_1_16.csv")
s1117 = pd.read_csv("s1_1_17.csv")
s1118 = pd.read_csv("s1_1_18.csv")

#IDENTIFICACIÓN DE CADA ARCHIVO
#SUJETO 1
s111['Subject'] = 1
s112['Subject'] = 1
s113['Subject'] = 1
```

```
s114['Subject'] = 1
s115['Subject'] = 1
s116['Subject'] = 1
s117['Subject'] = 1
s118['Subject'] = 1
s119['Subject'] = 1
s1110['Subject'] = 1
s1111['Subject'] = 1
s1112['Subject'] = 1
s1113['Subject'] = 1
s1114['Subject'] = 1
s1115['Subject'] = 1
s1116['Subject'] = 1
s1117['Subject'] = 1
s1118['Subject'] = 1
#_____________________________________________
#PRUEBA 1
s111['Test'] = 1
s112['Test'] = 1
s113['Test'] = 1
s114['Test'] = 1
s115['Test'] = 1
s116['Test'] = 1
s117['Test'] = 1
s118['Test'] = 1
s119['Test'] = 1
s1110['Test'] = 1
s1111['Test'] = 1
s1112['Test'] = 1
s1113['Test'] = 1
s1114['Test'] = 1
s1115['Test'] = 1
s1116['Test'] = 1
s1117['Test'] = 1
s1118['Test'] = 1
#_____________________________________________
```

```
#ENSAYOS
s111['Trial'] = 1
s112['Trial'] = 2
s113['Trial'] = 3
s114['Trial'] = 4
s115['Trial'] = 5
s116['Trial'] = 6
s117['Trial'] = 7
s118['Trial'] = 8
s119['Trial'] = 9
s1110['Trial'] = 10
s1111['Trial'] = 11
s1112['Trial'] = 12
s1113['Trial'] = 13
s1114['Trial'] = 14
s1115['Trial'] = 15
s1116['Trial'] = 16
s1117['Trial'] = 17
s1118['Trial'] = 18

#FUNCIONES
def time_column_in(df):
    t_col = []

    df = df[df['Conectivity']==1]

    df = df[df.isnull()['EEG_TP10']!=True]
    df = df[df.isnull()['EEG_AF8']!=True]
    df = df[df.isnull()['EEG_AF7']!=True]
    df = df[df.isnull()['EEG_TP9']!=True]

    shape = df.shape[0]

    for x in range(shape):
        t_col.append(x * 1/256)

    df['Time'] = t_col
    df['Sampling_Rate'] = 256

    return df
```

```
#IMPLEMENTACIÓN DE LA FUNCIÓN
s111 = time_column_in(s111)
s112 = time_column_in(s112)
s113 = time_column_in(s113)
s114 = time_column_in(s114)
s115 = time_column_in(s115)
s116 = time_column_in(s116)
s117 = time_column_in(s117)
s118 = time_column_in(s118)
s119 = time_column_in(s119)
s1110 = time_column_in(s1110)
s1111 = time_column_in(s1111)
s1112 = time_column_in(s1112)
s1113 = time_column_in(s1113)
s1114 = time_column_in(s1114)
s1115 = time_column_in(s1115)
s1116 = time_column_in(s1116)
s1117 = time_column_in(s1117)
s1118 = time_column_in(s1118)

#AGRUPACIÓN DE ENSAYOS
#TRAIN
db_1_train = pd.concat([s111, s112])
db_1_train = pd.concat([db_1_train, s113])
db_1_train = pd.concat([db_1_train, s114])
db_1_train = pd.concat([db_1_train, s115])
db_1_train = pd.concat([db_1_train, s116])
db_1_train = pd.concat([db_1_train, s117])
db_1_train = pd.concat([db_1_train, s118])
db_1_train = pd.concat([db_1_train, s119])
db_1_train = pd.concat([db_1_train, s1110])
db_1_train = pd.concat([db_1_train, s1111])
db_1_train = pd.concat([db_1_train, s1112])
db_1_train = pd.concat([db_1_train, s1113])
db_1_train = pd.concat([db_1_train, s1114])
db_1_train = pd.concat([db_1_train, s1115])
db_1_train = pd.concat([db_1_train, s1116])
```

```
db_1_train = db_1_train.reset_index(drop=True)

#TEST
db_1_test = pd.concat([s1117, s1118])

db_1_test = db_1_test.reset_index(drop=True)

#Comprobación de si quedan datos NaN
db_1_train.isnull().sum()

db_1_test.isnull().sum()

#Generar "Databases"
db_1_train.to_csv('db_1_train.csv')
db_1_test.to_csv('db_1_test.csv')
```

1.2.2. Procesado de Datos y Clasificación

```
#Instalar las librerías no incluidas en Google Colab
!pip install mne
!pip install python-picard

#Implementación de las librerías necesarias para el procesado
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

from picard import picard

import mne
from mne.preprocessing import ICA
from mne.io import RawArray
from mne.epochs import concatenate_epochs
from mne import create_info, find_events, Epochs, pick_types
from mne.decoding import CSP

from scipy.signal import kaiserord, firwin, freqz

from scipy.signal import butter, lfilter, convolve, boxcar
```

```
import math

#Permitir lectura de archivos guardados en drive
from google.colab import drive
drive.mount('/content/drive')

#Lectura de la base de datos de entrenamiento con pandas.
TrainDb_s1 = pd.read_csv('/content/drive/My Drive/Bases de datos TFG/db_1_train.csv')

#Se retira toda columna que no contenga los datos en bruto o las etiquetas.
TrainDb_s1 = TrainDb_s1.drop('Unnamed: 0',1)
TrainDb_s1 = TrainDb_s1.drop('Conectivity',1)
TrainDb_s1 = TrainDb_s1.drop('Subject',1)
TrainDb_s1 = TrainDb_s1.drop('Test',1)
TrainDb_s1 = TrainDb_s1.drop('Trial',1)
TrainDb_s1 = TrainDb_s1.drop('Time',1)
TrainDb_s1 = TrainDb_s1.drop('Sampling_Rate',1)

#Cambio de etiquetas en formato "String" por números enteros
#None ---> 0
#Izquierda ---> 1
#Derecha ---> 2
TrainDb_s1.Label[TrainDb_s1.Label=='None']=0
TrainDb_s1.Label[TrainDb_s1.Label=='Left']=1
TrainDb_s1.Label[TrainDb_s1.Label=='Right']=2

#Definición de la función encargada de convertir la base de datos en un objeto "RAW",
#es decir, el tipo de objeto que necesita la librería MNE para procesar sus datos.
def create_mne_raw_object(data):
    """Create a mne raw instance from csv file"""

    # Registro del nombre de los canales
    ch_names = list(data.columns[:4])
    ch_type = ['eeg']*len(ch_names)

    # Registro los datos en bruto habiéndolos pasado a la unidad de VOLTIOS
    values = 1e-6*np.array(data[ch_names]).T


      # Registro de eventos
    events_names = data.columns[4]
```

```
    events_data = np.array(data[events_names]).T
    events_data = events_data.reshape(1,events_data.shape[0])

    # definición del tipo de canal "stim" para identificar las etiquetas correspondientes
    # a cada evento
    ch_type.extend(['stim'])
    ch_names.extend([events_names])

    # Concatenar eventos y datos
    data = np.concatenate((values,events_data))

    # Crear una estructura de información MNE
    info = create_info(ch_names,sfreq=256, ch_types=ch_type)

    # Crear objecto raw
    raw = RawArray(data,info)

    return raw

#Aplicar función
raw_TrainDb_s1 = create_mne_raw_object(TrainDb_s1)

#Crear array con el conjunto de canales
picks = pick_types(raw_TrainDb_s1.info, meg=False, eeg=True, stim=False, eog=False)

#Impresión de las señales EEG en bruto a través de la librería "MNE"
raw_TrainDb_s1.plot(duration=25, start=70)

#Impresión de la señal EEG en bruto en el canal TP9 a través de la librería "Matplotlib"
fs = 256
t = np.arange(raw_TrainDb_s1._data.shape[1]) / fs

plt.figure(0)
plt.plot(t, raw_TrainDb_s1._data[3], 'g', label="Canal TP9")
plt.xlabel('t')
plt.ylabel('uV')
plt.legend(loc="lower right", frameon=False)
plt.grid(True)

#Impresión del espectro de potencia a través de la librería "MNE"
raw_TrainDb_s1.plot_psd(tmin=70, tmax=72, fmax=60, average=True)
```

```
# Diseño de filtro FIR paso banda ---> 0.5 a 50 Hz
#Frecuencia de Muestreo
fs = 256

#Frecuencia de Nyquist:
f_nyq  = fs/ 2.0

#Frecuencias de corte:
freqs = [0.5,50]

#Número de taps
N = 900

# Diseño del filtro por medio de una ventana hamming
taps = firwin(N, np.array(freqs)/f_nyq, window='hamming', pass_zero = 'bandpass')

#Imprimir Coeficientes del filtro FIR con "Matplotlib"
plt.figure(1)
plt.plot(taps, linewidth=2)
plt.title('Filter Coefficients (%d taps)' % N)
plt.grid(True)

#Imprimir magnitud de respuesta del filtro con "Matplotlib"
plt.figure(2)
w, h = freqz(taps)
plt.plot((w/math.pi)*f_nyq, np.absolute(h), linewidth=2)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Gain')
plt.title('Frequency Response')
plt.grid(True)

#IMPRIMIR RESULTADOS TRAS EL FILTRADO EN TP9
t = np.arange(raw_TrainDb_s1._data.shape[1]) / fs

# Cálculo del retardo de fase de la señal filtrada
delay = 0.5 * (N-1) / fs

#Señal original
plt.figure(3)

plt.plot(t, raw_TrainDb_s1._data[3], 'g', label = "Original TP9 Signal")
```

```
#Señal filtrada sin omitir el retardo de fase
plt.plot(t-delay, lfilter(taps, 1.0, raw_TrainDb_s1._data[3]),
        'r-', label = "Phase Delay")

#Señal filtrada y sin retardo de fase
plt.plot(t[N-1:]-delay, lfilter(taps, 1.0, raw_TrainDb_s1._data[3])
        [N-1:], label = "Filtered TP9 Signal")
plt.legend(loc="lower right", frameon=False)
plt.xlabel('t')
plt.grid(True)

# Aplicacion del filtro sobre la base de datos
raw_TrainDb_s1._data[picks] = lfilter(taps, 1.0, raw_TrainDb_s1._data[picks])

#Eliminación del retardo de fase
shape = raw_TrainDb_s1._data.shape[1]
labels = raw_TrainDb_s1._data[4,:(shape-int((N)/2))]
shape = labels.shape[0]
labels = np.reshape(labels,[1,shape])
labels = labels[0,(int((N-1)/2)):]
shape = labels.shape[0]
labels = np.reshape(labels,[1,shape])

raw_TrainDb_s1._data = raw_TrainDb_s1._data[picks,(N-1):]
raw_TrainDb_s1._data = np.append(raw_TrainDb_s1._data, labels, axis = 0)

#Impresión de las señales EEG filtradas a través de la librería "MNE"
raw_TrainDb_s1.plot(duration=25, start=70)

#Impresión del espectro de potencia de la señal filtrada a través de la librería "MNE"
raw_TrainDb_s1.plot_psd(tmin=70, tmax=72, fmax=60, average=True)

#Prueba de detección de artefactos EOG
eog_epochs = mne.preprocessing.create_eog_epochs(raw_TrainDb_s1,
        ch_name='EEG_TP9', tmin=-0.1, tmax=0.4)
eog_epochs.plot_image(combine='mean')

#Cargar algoritmo de ICA
ica = ICA(method='picard')

#Aplicar algoritmo para extraer los ICs
ica.fit(raw_TrainDb_s1)
```

```
#Imprimir ICs
raw_TrainDb_s1.load_data()
ica.plot_sources(raw_TrainDb_s1, start=70, stop=95)

#Imprimir señales en caso de retirar el IC "ICA000"
ica.plot_overlay(raw_TrainDb_s1, exclude=[0], picks='eeg', start=70*256, stop=95*256)

#Retirada del IC "ICA000"
ica.exclude = [0]
ica.apply(raw_TrainDb_s1)

#Impresión de las señales EEG sin ICA000
raw_TrainDb_s1.plot(duration=25, start=70)

#Diseño de filtro FIR para filtrar los datos exluyendo la banda beta ---> 13 y 30 Hz
#Frecuencia de Muestreo
fs = 256

#Frecuencia de Nyquist:
f_nyq  = fs/ 2.0

#Frecuencias de corte:
freqs = [13,30]

#Número de taps
N = 200

# Diseño del filtro por medio de una ventana hamming
taps_2 = firwin(N, np.array(freqs)/f_nyq, window='hamming', pass_zero = 'bandpass')

#Imprimir Coeficientes del filtro FIR con "Matplotlib"
plt.figure(4)
plt.plot(taps_2, linewidth=2)
plt.title('Filter Coefficients (%d taps)' % N)
plt.xlabel('Frequency (Hz)')
plt.grid(True)

#Imprimir magnitud de respuesta del filtro con "Matplotlib"
plt.figure(5)
w, h = freqz(taps_2)
plt.plot((w/math.pi)*f_nyq, np.absolute(h), linewidth=2)
plt.xlabel('Frequency (Hz)')
```

```
plt.ylabel('Gain')
plt.title('Frequency Response')
plt.grid(True)

#IMPRIMIR RESULTADOS TRAS EL FILTRADO EN TP9
t = np.arange(raw_TrainDb_s1._data.shape[1]) / fs

#Cálculo del retardo de fase de la señal filtrada
delay = 0.5 * (N-1) / fs

#Señal original
plt.figure(6)

plt.plot(t, raw_TrainDb_s1._data[3], 'g', label = "Original TP9 Signal")

#Señal filtrada sin omitir el retardo de fase
plt.plot(t-delay, lfilter(taps_2, 1.0, raw_TrainDb_s1._data[3]),
        'r-', label = "Phase Delay")

# Señal filtrada y sin retardo de fase
plt.plot(t[N-1:]-delay, lfilter(taps_2, 1.0, raw_TrainDb_s1._data[3])
        [N-1:], label = "Filtered TP9 Signal")
plt.legend(loc="upper right", frameon=False)
plt.xlabel('t')
plt.grid(True)

# Aplicacion del filtro sobre la base de datos
raw_TrainDb_s1._data[picks] = lfilter(taps_2, 1.0, raw_TrainDb_s1._data[picks])

#Eliminación del retardo de fase
shape = raw_TrainDb_s1._data.shape[1]
labels = raw_TrainDb_s1._data[4,:(shape-int((N)/2))]
shape = labels.shape[0]
labels = np.reshape(labels,[1,shape])
labels = labels[0,(int((N-1)/2)):]
shape = labels.shape[0]
labels = np.reshape(labels,[1,shape])

raw_TrainDb_s1._data = raw_TrainDb_s1._data[picks,(N-1):]
raw_TrainDb_s1._data = np.append(raw_TrainDb_s1._data, labels, axis = 0)

#Espectro de potencia tras haber aplicado el filtro
raw_TrainDb_s1.plot_psd(tmin=20, tmax=22, fmax=60, average=True)
```

```
#Impresión de las señales EEG filtradas
raw_TrainDb_s1.plot(duration=25, start=500)

#Retirada de canales AF8 y TP10
raw_TrainDb_s1.drop_channels(['EEG_AF8'])
raw_TrainDb_s1.drop_channels(['EEG_TP10'])

picks = pick_types(raw_TrainDb_s1.info, meg=False, eeg=True, stim=False, eog=False)

#ENTRENAMIENTO DE LOS FILTROS CSP
events = find_events(raw_TrainDb_s1)

# Señal entre 0.5 y 2.5 segundos tras ocurrir el evento 1 (Left)
epochs = Epochs(raw_TrainDb_s1, events, {'during left' : 1}, 0.5, 2.5, proj=False,
picks=picks, baseline=None, preload=True, verbose=False)

epochs_tot = []
y = []

epochs_tot.append(epochs)
y.extend([1]*len(epochs))

# Señal entre 0.5 y 2.5 segundos tras ocurrir el evento 2 (Right)
epochs_rest = Epochs(raw_TrainDb_s1, events, {'during right' : 2}, 0.5, 2.5,
      proj=False, picks=picks, baseline=None, preload=True, verbose=False)

y.extend([-1]*len(epochs_rest))
epochs_tot.append(epochs_rest)

#Agrupar el conjunto de "epochs"
epochs = concatenate_epochs(epochs_tot)

#Conjunto de datos para entrenar el algoritmo CSP
X = epochs.get_data()

#Conjunto de etiquetas para entrenar el algoritmo CSP
y = np.array(y)

#Selección del número de filtros
nfilters = 2

#Entrenamiento de filtros
csp = CSP(n_components=nfilters, reg='ledoit_wolf')
csp.fit(X,y)
```

```
#Retirada de los datos relacioados con la etiqueta "None"
df = pd.DataFrame(data=raw_TrainDb_s1._data.T)
df = df[df[2]!=0]
raw_TrainDb_s1._data = df.to_numpy().T

# Aplicar filtros  sobre el conjunto total e datos y rectificado de las señales resultantes
feat_train = np.dot(csp.filters_[0:nfilters],raw_TrainDb_s1._data[picks])**2

#Impresión de los resultados obtenidos
t = np.arange(raw_TrainDb_s1._data.shape[1]) / fs

#Canal AF7
plt.figure(7)
plt.plot(t, feat_train[0], 'g', label = "Signal 1")
plt.plot(t, raw_TrainDb_s1._data[2], 'r', label = "Labels")
plt.xlabel('t')
plt.xlim(180, 220)
plt.ylim(0, 10)
plt.legend(loc="upper right", frameon=False)
plt.grid(True)

#Canal TP9
plt.figure(8)
plt.plot(t, feat_train [1], 'b', label = "Signal 2")
plt.plot(t, raw_TrainDb_s1._data[2], 'r', label = "Labels")
plt.legend(loc="upper right", frameon=False)
plt.xlabel('t')
plt.xlim(180, 220)
plt.ylim(0, 10)
plt.grid(True)

#Alisado de la señal con ventanas de 800 muestras
nwin = 800

train_f = [convolve(feat_train[i],boxcar(nwin),'full') for i in range(nfilters)]
train_f = np.array(train_f)
train_f = np.log(train_f [:,0:feat_train.shape[1]])

#Array con las etiquetas correspondientes a los datos resultantes
labels_train = raw_TrainDb_s1._data[-1]
```

```
#Renumerado de las etiquetas
#Left ---> 0
#Right --> 1
for i in range(labels_train.size):
    if labels_train[i] == 1:
        labels_train[i] = 0

    else:
        labels_train[i] = 1

#Impresión de las señales resultantes del procesado sobre el database Train
t = np.arange(train_f.shape[1]) / fs

#Canal AF7
plt.figure(9)
plt.plot(t, train_f [0], 'g', label = "Signal 1")
plt.plot(t, labels_train, 'r', label = "Labels")
plt.xlabel('t')
plt.xlim(180, 210)
plt.legend(loc="lower right", frameon=False)
plt.grid(True)

#Canal TP9
plt.figure(10)
plt.plot(t, train_f [1], 'b', label = "Signal 2")
plt.plot(t, labels_train, 'r', label = "Labels")
plt.xlabel('t')
plt.xlim(180, 210)
plt.legend(loc="lower right", frameon=False)
plt.grid(True)

#Lectura de la base de datos de Test
TestDb_s1 = pd.read_csv('/content/drive/My Drive/Bases de datos TFG/db_1_test.csv')

#Mismo procesado que con el database de Train (sin rediseñar los algoritmos)
TestDb_s1 = TestDb_s1.drop('Unnamed: 0',1)
TestDb_s1 = TestDb_s1.drop('Conectivity',1)
TestDb_s1 = TestDb_s1.drop('Subject',1)
TestDb_s1 = TestDb_s1.drop('Test',1)
TestDb_s1 = TestDb_s1.drop('Trial',1)
```

```
TestDb_s1 = TestDb_s1.drop('Time',1)
TestDb_s1 = TestDb_s1.drop('Sampling_Rate',1)

TestDb_s1.Label[TestDb_s1.Label=='None']=0
TestDb_s1.Label[TestDb_s1.Label=='Left']=1
TestDb_s1.Label[TestDb_s1.Label=='Right']=2

raw_TestDb_s1 = create_mne_raw_object(TestDb_s1)

picks = pick_types(raw_TestDb_s1.info, meg=False, eeg=True, stim=False, eog=False)

N = 900
raw_TestDb_s1._data[picks] = lfilter(taps, 1.0, raw_TestDb_s1._data[picks])

shape = raw_TestDb_s1._data.shape[1]
labels = raw_TestDb_s1._data[4,:(shape-(N-1))]
shape = labels.shape[0]
labels = np.reshape(labels,[1,shape])

raw_TestDb_s1._data = raw_TestDb_s1._data[picks,N-1:]

raw_TestDb_s1._data = np.append(raw_TestDb_s1._data, labels, axis = 0)

ica.exclude = [0]
ica.apply(raw_TestDb_s1)

N = 200
raw_TestDb_s1._data[picks] = lfilter(taps_2, 1.0, raw_TestDb_s1._data[picks])

shape = raw_TestDb_s1._data.shape[1]
labels = raw_TestDb_s1._data[4,:(shape-(N-1))]
shape = labels.shape[0]
labels = np.reshape(labels,[1,shape])

raw_TestDb_s1._data = raw_TestDb_s1._data[picks,N-1:]

raw_TestDb_s1._data = np.append(raw_TestDb_s1._data, labels, axis = 0)

raw_TestDb_s1.drop_channels(['EEG_AF8'])
raw_TestDb_s1.drop_channels(['EEG_TP10'])

picks = pick_types(raw_TrainDb_s1.info, meg=False, eeg=True, stim=False, eog=False)

df = pd.DataFrame(data=raw_TestDb_s1._data.T)
df = df[df[2]!=0]
```

```
raw_TestDb_s1._data = df.to_numpy().T

feat_test = np.dot(csp.filters_[0:nfilters],raw_TestDb_s1._data[picks])**2

test_f = [convolve(feat_test[i],boxcar(nwin),'full') for i in range(nfilters)]
test_f = np.array(test_f)
test_f = np.log(test_f [:,0:feat_test.shape[1]])

labels_test = raw_TestDb_s1._data[-1]

for i in range(labels_test.size):
    if labels_test[i] == 1:
        labels_test[i] = 0

    else:
        labels_test[i] = 1

#Impresión de las señales resultantes
t = np.arange(test_f.shape[1]) / fs

#Canal AF7
plt.figure(11)
plt.plot(t, test_f [0], 'g', label = "Signal 1")
plt.plot(t, labels_test, 'r', label = "Labels")
plt.xlabel('t')
plt.xlim(110, 150)
plt.legend(loc="lower right", frameon=False)
plt.grid(True)

#Canal TP9
plt.figure(12)
plt.plot(t, test_f [1], 'b', label = "Signal 2")
plt.plot(t, labels_test, 'r', label = "Labels")
plt.xlabel('t')
plt.xlim(110, 150)
plt.legend(loc="lower right", frameon=False)
plt.grid(True)

#Impresión de la distribución de etiquetas en Train
plt.hist(labels_train)
plt.text(1.1, 300000, "Left --> 0",fontsize=14)
```

```
plt.text(1.1, 260000, "Right --> 1",fontsize=14)

#Impresión de la distribución de etiquetas en Test
plt.hist(labels_test)
plt.text(1.1, 35000, "Left --> 0",fontsize=14)
plt.text(1.1, 30000, "Right --> 1",fontsize=14)

#CLASIFICACIÓN DE LAS CARACTERÍSTICAS
#Implementar librerías
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score

#Entrenar modelo con el algoritmo de Regresión Logística
lr = LogisticRegression()
lr.fit(train_f.T, labels_train)

# Clasificar características de la base de datos de Test
value_pred = lr.predict(test_f.T)
predictions = [round(val) for val in value_pred]

labels_test = raw_TestDb_s1._data[-1]

# Evaluar los resultados
accuracy = accuracy_score(labels_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

#Matriz de confusión
matrix = confusion_matrix(labels_test, predictions)
matrix

#Impresión de la distribución de las predicciones
plt.hist(labels_test)
plt.hist(predictions)

#Clasificar características de la base de datos de Train
value_pred = lr.predict(train_f.T)
predictions = [round(val) for val in value_pred]

labels_train = raw_TrainDb_s1._data[-1]
```

```
#Evaluar los resultados
accuracy = accuracy_score(labels_train, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

#Matriz de confusión
matrix = confusion_matrix(labels_train, predictions)
print(matrix)

#Impresión de la distribución de las predicciones
plt.hist(labels_train)
plt.hist(predictions)

#Entrenar modelo con el algoritmo de XGboost
xgb = XGBClassifier(tree_method='exact')
xgb.fit(train_f.T, labels_train)

# Clasificar características de la base de datos de Test
value_pred = lr.predict(test_f.T)
predictions = [round(val) for val in value_pred]

labels_test = raw_TestDb_s1._data[-1]

# Evaluar los resultados
accuracy = accuracy_score(labels_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

#Matriz de confusión
matrix = confusion_matrix(labels_test, predictions)
matrix

#Impresión de la distribución de las predicciones
plt.hist(labels_test)
plt.hist(predictions)

#Clasificar características de la base de datos de Train
value_pred = xgb.predict(train_f.T)
predictions = [round(val) for val in value_pred]

labels_train = raw_TrainDb_s1._data[-1]

#Evaluar los resultados
```

```
accuracy = accuracy_score(labels_train, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

#Matriz de confusión
matrix = confusion_matrix(labels_train, predictions)
print(matrix)

#Impresión de la distribución de las predicciones
plt.hist(labels_train)
plt.hist(predictions)
```

2. MIQ-R

CUESTIONARIO REVISADO DE IMAGEN DEL MOVIMIENTO (MIQ-R) [1]

Hall y Martin

Instrucciones

Este cuestionario incluye dos formas de lleva a cabo movimientos mentalmente. Algunas personas utilizan más una forma que otra, y algún tipo de forma mental se aplica más a unos movimientos que a otros.

Lo primero que tienes que hacer es formar una imagen visual o dibujo de un movimiento en tu mente. Y, lo segundo, es intentar sentir la realización de un movimiento como si se estuviese haciendo. Se pide que hagas esas dos tareas mentales cuando en el cuestionario se describan movimientos.

Debes valorar lo fácil o difícil que te resulta la realización mental de esos movimientos. No se pretende evaluar lo bueno o malo que eres efectuando tareas mentales, sino que lo que se intenta es descubrir la capacidad de los individuos para realizar esas tareas para diferentes movimientos. No existen valoraciones buenas o malas, y no son mejores unas que otras.

Cada una de las siguientes exposiciones describen una acción o movimiento concreto. Lee cada uno de los planteamientos con cuidado y después realiza realmente el movimiento que se ha descrito. Realiza el movimiento solamente una vez. Vuelve a la posición inicial, como si fueses a intentar el movimiento una segunda vez. Después, depende de lo que a continuación se te pida hacer, bien 1) formar una imagen mental tan clara y viva como te sea posible del movimiento que acabas de realizar, o 2) intentar sentir el movimiento que acabas de realizar, pero sin ejecutarlo realmente.

Después de haber completado la tarea mental requerida, valora la facilidad/dificultad con la que has podido hacer la tarea. Realiza la valoración siguiendo la escala que se te presenta a continuación. Sé tan preciso como sea posible, y utiliza tanto tiempo como creas necesario para llegar a la valoración apropiada para cada movimiento. Puedes elegir la misma valoración para cualquier número de movimientos "imaginados" o "sentidos", y no es necesario utilizar todas las valoraciones de la escala.

MIQ-R

Escalas de valoración

Escala de Imagen Visual	*Puntuación*
Muy fácil de formar la imagen	7
Fácil de formar la imagen	6
Algo fácil de formar la imagen	5
Neutral (ni fácil, ni difícil)	4
Algo difícil de formar la imagen	3
Difícil de formar la imagen	2
Muy difícil de formar la imagen	1

Escala de Imagen Cinestésica	*Puntuación*
Muy fácil de formar la imagen	7
Fácil de formar la imagen	6
Algo fácil de formar la imagen	5
Neutral (ni fácil, ni difícil)	4
Algo difícil de formar la imagen	3
Difícil de formar la imagen	2
Muy difícil de formar la imagen	1

1.- POSICION INICIAL: Colócate con tus pies y piernas juntos y tus brazos en los costados.

EJERCICIO: Levanta tu rodilla derecha tan alto como sea posible, de tal modo que estés recto sobre tu pierna izquierda con tu pierna derecha flexionada (doblada) a la altura de la rodilla. Ahora baja tu pierna derecha, de tal modo que estés de nuevo de pie sobre los dos pies. Ejecuta estas acciones lentamente.

TAREA MENTAL: Adopta la posición inicial. Intenta sentir que tú estás haciendo el movimiento que acabas de realizar, pero sin ejecutarlo realmente. Ahora, valora la facilidad/dificultad con la que has podido hacer esta tarea mental.

Valoración.....

2.- POSICION INCIAL: Colócate con tus pies un poco separados y tus manos en los costados.

EJERCICIO: Inclínate hacia abajo y después salta, poniéndote de pie en el aire, tan alto como sea posible, con ambos brazos extendidos por encima de tu cabeza. Cae con tus pies separados y baja tus brazos a los costados.

TAREA MENTAL: Adopta la posición inicial. Forma una imagen mental tan clara y viva como te sea posible del movimiento que acabas de realizar. Ahora, valora la facilidad/dificultad con la que has podido hacer esta tarea mental.

Valoración……

3.- POSICION INICIAL: Extiende el brazo de tu mano no dominante totalmente recto en dirección lateral, de tal modo que esté paralelo al suelo, con la palma hacia abajo.

EJERCICIO: Mueve tu brazo hacia delante hasta que esté directamente enfrente de tu cuerpo (todavía paralelo al suelo). Mantén tu brazo extendido durante el movimiento y haz el movimiento lentamente.

TAREA MENTAL: Adopta la posición inicial. Intenta sentir que tú estás haciendo el movimiento que acabas de realizar, pero sin ejecutarlo realmente. Ahora, valora la facilidad/dificultad con la que has podido hacer esta tarea mental.

Valoración……

4.- POSICION INICIAL: Colócate con tus pies un poco separados y tus brazos completamente extendidos encima de tu cabeza.

EJERCICIO: Dobla lentamente hacia delante tu cintura e intenta tocar la punta de tus pies con la yema de tus dedos (o si es posible, tocar el suelo con la yema de tus dedos o con las manos). Ahora vuelve a la posición inicial, de pie, recto, con tus brazos extendidos encima de tu cabeza.

TAREA MENTAL: Adopta la posición inicial. Forma una imagen mental tan clara y viva como te sea posible del movimiento que acabas de realizar. Ahora, valora la facilidad/dificultad con la que has podido hacer esta tarea mental.

Valoración……

5.- POSICION INICIAL: Colócate con tus pies un poco separados y tus manos en los costados.

EJERCICIO: Inclínate hacia abajo y después salta, poniéndote de pie en el aire, tan alto como te sea posible, con ambos brazos extendidos por encima de tu cabeza. Cae con tus pies separados y baja tus brazos a los costados.

TAREA MENTAL: Adopta la posición inicial. Intenta sentir que tú estás haciendo el movimiento que acabas de realizar, pero sin ejecutarlo realmente. Ahora, valora la facilidad/dificultad con la que has podido hacer esta tarea mental.

Valoración……

6.- POSICION INICIAL: Colócate con tus pies y piernas juntos y tus brazos en los costados.

EJERCICIO: Levanta tu rodilla derecha tan alto como sea posible, de tal modo que estés recto sobre tu pierna izquierda, y tu pierna derecha flexionada (doblada) en la rodilla. Ahora baja tu pierna derecha, de tal modo que estés de nuevo de pie sobre los dos pies. Ejecuta estas acciones lentamente.

TAREA MENTAL: Adopta la posición inicial. Forma una imagen mental tan clara y viva como te sea posible del movimiento que acabas de realizar. Ahora, valora la facilidad/dificultad con la que has podido hacer esta tarea mental.

Valoración……

7.- POSICION INICIAL: Colócate con tus pies un poco separados y tus brazos completamente extendidos encima de tu cabeza.

EJERCICIO: Dobla lentamente hacia delante tu cintura e intenta tocar la punta de tus pies con la yema de tus dedos (o si es posible, tocar el suelo con la yema de tus dedos o con las manos). Ahora vuelve a la posición inicial, de pie, recto, con tus brazos extendidos encima de tu cabeza.

TAREA MENTAL: Adopta la posición inicial. Intenta sentir que tú estás haciendo el movimiento que acabas de realizar, pero sin ejecutarlo realmente. Ahora, valora la facilidad/dificultad con la que has podido hacer esta tarea mental.

Valoración……

8.- POSICION INICIAL: Extiende el brazo de tu mano no dominante totalmente recto en dirección lateral, de tal modo que esté paralelo al suelo, con la palma hacia abajo.

EJERCICIO: Mueve tu brazo hacia delante hasta que esté directamente enfrente de tu cuerpo (todavía paralelo al suelo). Mantén tu brazo extendido durante el movimiento y haz el movimiento lentamente.

TAREA MENTAL: Adopta la posición inicial. Forma una imagen mental tan clara y viva como te sea posible del movimiento que acabas de realizar. Ahora, valora la facilidad/dificultad con la que has podido hacer esta tarea mental.

Valoración……

3. BIBLIOGRAFÍA

[1] A. Campos y M. A. González, «Versión Española del Cuestionario-Revisado de Imagen del Movimiento (MIQ-R): Validación y Propiedades Psicométricas», p. 13.

Relación de documentos

(_) Memoria	89	páginas
(X) Anexos	46	páginas

La Almunia, a 25 de noviembre de 2020

Firmado: Iñigo Garaboa Cotelo