```python
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

#%%
#                                                    "TEST ON THE INDEX OF DATA"
UDP_PORT = 49000

import socket
import struct
import datetime
import pandas as pd
import csv


counter = 0

def DecodeDataMessage(message):
  # Message consists of 4 byte type and 8 times a 4byte float value.
  # Write the results in a python dict.
  values = {}
  typelen = 4
  type = int.from_bytes(message[0:typelen], byteorder='little')
  data = message[typelen:]
  dataFLOATS = struct.unpack("<ffffffff",data)

  if type == 1:

      global counter
      counter = counter + 1

      currentDT = datetime.datetime.now()
      stamp = currentDT.strftime('%m/%d/%Y %H:%M:%S.%f')

      values['timestamp'] = stamp
      values['real, time']=dataFLOATS[0]
      values["totl, time"]=dataFLOATS[1]
      #values["missn, time"]=dataFLOATS[2]
```

```
    #values["timer, time"]=dataFLOATS[3]
    #values["zulu, time"]=dataFLOATS[4]
    #values["local, time"]=dataFLOATS[5]
    #values["hobbs, time"]=dataFLOATS[6]

elif type == 3:
    values['Vind, kias']=dataFLOATS[0] # airspeed indicator
    #values["Vind, keas"]=dataFLOATS[1]
    #values["Vtrue, ktas"]=dataFLOATS[2]
    #values["Vtrue, ktgs"]=dataFLOATS[3]
    #values["Vind, mph"]=dataFLOATS[4]
    #values["Vtrue, mphas"]=dataFLOATS[5]
    #values["Vtrue, mphgs"]=dataFLOATS[6]

elif type == 4:
    #values["Mach, ratio"]=dataFLOATS[0]
    values['VVI, fpm']=dataFLOATS[2] # VSI indicator
    #values["Gload, norml"]=dataFLOATS[2]
    #values["Gload, axial"]=dataFLOATS[3]
    #values["Gload, side"]=dataFLOATS[4]

elif type == 8:
    values['elev, yoke1']=dataFLOATS[0] # pull +1 , push -1
    values['ailrn, yoke1']=dataFLOATS[1] # right +1, left -1
    values['ruddr, yoke1']=dataFLOATS[2] # rudder right +1, left -1

elif type == 13:
    values['trim, elev']=dataFLOATS[0] # showes when the user apply pitch trim
    values['trim, ailrn']=dataFLOATS[1] # showes when the users apply aileron trim
    #values["trim, rudder"]=dataFLOATS[2]
    values['flap, handl']=dataFLOATS[3] # flap position,  0, 0.333, 1
    #values["flap position"]=dataFLOATS[4]
    #values["slat, ratio"]=dataFLOATS[5]

elif type == 17:
    values['pitch, deg']=dataFLOATS[0] # pitch deg - atitude indicator
    values['roll, deg']=dataFLOATS[1] # roll degree - atitude indicator
    #values["heading, true"]=dataFLOATS[2]
    values["heading mag"]=dataFLOATS[3] # heading indicator

elif type == 20:
```

```
        #values["lat, deg"]=dataFLOATS[0]
        #values["lon, deg"]=dataFLOATS[1]
        #values["alt, MSL"]=dataFLOATS[2]
        values['alt, ind']=dataFLOATS[5] # altitude indicator

    elif type == 21:
        values['X, m']=dataFLOATS[0] # X
        values['Y, m']=dataFLOATS[1] # Y
        values['Z, m']=dataFLOATS[2] # Z
        #values["vX, m/s"]=dataFLOATS[3]
        #values["vY, m/s"]=dataFLOATS[4]
        #values["vZ, m/s"]=dataFLOATS[5]

    elif type == 37:
        values['rpm n, engin']=dataFLOATS[0] # RPM

    #else:
        #print("  Type ", type, " not implemented: ",dataFLOATS)
    return values

def DecodePacket(data):
    # Packet consists of 5 byte header and multiple messages.
    valuesout = {}
    headerlen = 5
    header = data[0:headerlen]
    messages = data[headerlen:]
    if(header==b'DATA*'):
        # Divide into 36 byte messages
        messagelen = 36
        for i in range(0,int((len(messages))/messagelen)):
            message = messages[(i*messagelen) : ((i+1)*messagelen)]
            values = DecodeDataMessage(message)
            valuesout.update( values )
    else:
        print("Packet type not implemented. ")
        print("  Header: ", header)
        print("  Data: ", messages)
    return valuesout

def main():
```

```
# Open a Socket on UDP Port 49000
UDP_IP = '192.168.1.100'
sock = socket.socket(socket.AF_INET, # Internet
                     socket.SOCK_DGRAM) # UDP

sock.bind((UDP_IP, UDP_PORT))

while True:
  # Receive a packet
  data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes

  # Decode the packet. Result is a python dict (like a map in C) with values from X-Plane.
  # Example:
  # {'latitude': 47.72798156738281, 'longitude': 12.434000015258789,
  #   'altitude MSL': 1822.67, 'altitude AGL': 0.17, 'speed': 4.11,
  #   'roll': 1.05, 'pitch': -4.38, 'heading': 275.43, 'heading2': 271.84}
  values = DecodePacket(data)
  print(values['timestamp'])
  #(pd.DataFrame.from_dict(data=values, orient='index').to_csv('dict_file.csv', header=False))
  print (counter)

  #' N03-E-B-LDTS
  #' N03-E-A-LTDS
  #' N04-E-B-LTDS
  #' N04-E-A-LSDT
  #' N05-E-B-LTDS
  #' N05-E-A-TSDL
  #' N06-E-B-LTDS
  #' N06-E-A-LDTS

  with open('TEST-A.csv','a') as f1:
      writer=csv.writer(f1, delimiter='\t',lineterminator='\n')

      row =[]

      for key, value in values.items():

          if counter == 1:
              row.append(key)

          else:
```

```python
                row.append(value)

        writer.writerow(row)

if __name__ == '__main__':
  main()
#%%

#%% DISSERTATION ==> For EYE MOVEMENTS
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# this import data from tobii pro glass 2 as .tsv file.
# then create timestamps and interploate data on a 100 millisecond time interval
# * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

# VERSION 237
# THIS VERSION WORKS FOR ONE PARTICIPANTS, I GOT TO THE POINT THAT THIS IS NOT WORKING ANYMORE AND I NEED TO LOOK AT THE BIG PICTUR
# THIS HAS SEVERAL BENEFITS (1) I CAN WRAP UP THE STUDY AND HAVE A GOOD ESTIMATION OF WHAT HOW MUCH IS LEFT TO BE DONE (2) I CAN SU
# I HAVE TO DO TWO THINGS AT THE SAME TIME, FIRST TO HAVE A DATABASE OF COLLECTED DATA AND SECOND WORK ON INPUT DETECTION


# VERSION 324
# I REMOVED EXTRA AREAS OF INTERST FROM THE CODE AND ANALYSIS

import pandas as pd
import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib.colors
import matplotlib.style as style
import matplotlib.cm as cm
```

```
import matplotlib.ticker as mticker
from matplotlib import ticker

import datetime as dt
import time

import math

import statsmodels.api as sm
import glob

from timeit import Timer
from time import gmtime, strftime

from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.linear_model import LinearRegression
from sklearn import datasets
from sklearn.preprocessing import PolynomialFeatures

import scipy
from scipy import stats
from scipy.spatial.distance import cdist
from scipy.integrate import simps
from scipy.misc import electrocardiogram
from scipy.signal import find_peaks
from scipy.signal import argrelextrema

start_time = strftime("%Y-%m-%d %H:%M:%S")

t0 = time.time()

        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################
        ############################################################################ FOR DIVIDING MASTER FILES TO PARTICIPANTS FILES #######
        ##########################################################################################################################
        ##########################################################################################################################
```

```
        ##########################################################################################################################

dividing_file = 0

if dividing_file == 1:
    print('=================================== READING MASTER FILE OF EYE MOVEMENT ===================================')
    tobii = pd.read_csv('Tobii.tsv', encoding = 'utf-16', delimiter='\t', low_memory=False)

    print('=================================== DIVIDING MASTER FILE OF EYE MOVEMENT INTO FILES / PARTICIPANTS ==================
    tobii_recording_names = tobii['Participant name'].unique().tolist()
    tobii_recording_names

    name_counter = 0
    for name in tobii_recording_names:
        name_counter += 1
        tsv_flag = True

        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
        print(name_counter , '=====>', name)
        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++')

        divided_file = tobii.loc[(tobii['Participant name'] == name)]

        if name == 'N04-C-B-LDST':
            divided_file.loc[:, 'Participant name'] = 'N11-E-B-DSTL'
            name = 'N11-E-B-DSTL'

        elif name == 'N04-C-A-STDL':
            divided_file.loc[:, 'Participant name'] = 'N11-E-A-LSDT'
            name = 'N11-E-A-LSDT'

        elif name == 'N11-E-B-DSTL':
            divided_file.loc[:, 'Participant name'] = 'N04-C-B-LDST'
            name = 'N04-C-B-LDST'

        elif name == 'N11-E-A-LSDT':
            divided_file.loc[:, 'Participant name'] = 'N04-C-A-STDL'
            name = 'N04-C-A-STDL'

        elif name == 'N04-E-B-LTDS':
            divided_file.loc[:, 'Participant name'] = 'N05-C-A-STDL'
```

```
        name = 'N05-C-A-STDL-LT'

    elif name == 'N04-E-A-LSDT':
        divided_file.loc[:, 'Participant name'] = 'N07-E-A-LSTD'
        name = 'N07-E-A-LSTD-DE'

        ##############################################################################################
        # MODIFIYING THE EDITIED FILES OF EYE TRACKING
        ##############################################################################################
        # ============================================================================================
        # N01-C-A-LTSD-E
        # ============================================================================================
    elif name == 'N01-C-A-LTSD-E':
        print('FILE DELETED: ', name)
        tsv_flag = False
        continue
        # ============================================================================================
        # N01-C-A-LTSD-E
        # ============================================================================================
        # ============================================================================================
        # N02-C-A-DSTL-E
        # ============================================================================================
    elif name == 'N02-C-A-DSTL-E':
        print('FILE DELETED: ', name)
        tsv_flag = False
        continue
        # ============================================================================================
        # N02-C-A-DSTL-E
        # ============================================================================================
        # ============================================================================================
        # N02-C-A-DSTL-E
        # ============================================================================================
    elif name == 'N03-C-A-DSTL-E':
        divided_file.loc[:, 'Participant name'] = name[0:12]
        name = 'N03-C-A-DSTL-LF'
        # ============================================================================================
        # N02-C-A-DSTL-E
        # ============================================================================================

        # ============================================================================================
        # N07-E-A-LSTD
```

```
    # ================================================================================
elif name == 'N07-E-A-LSTD':
    divided_file.loc[:, 'Participant name'] = name[0:12]
    name = 'N07-E-A-LSTD-LA-LF'

elif name == 'N07-E-A-LSTD -2':
    divided_file.loc[:, 'Participant name'] = name[0:12]
    name = 'N07-E-A-LSTD-LT'

elif name == 'N07-E-A-LSTD -3':
    print('FILE DELETED: ', name)
    tsv_flag = False
    continue

elif name == 'N07-E-A-LSTD - E':
    print('FILE DELETED: ', name)
    tsv_flag = False
    continue
    # ================================================================================
    # N07-E-A-LSTD
    # ================================================================================
    # ================================================================================
    # N08-E-A-DTSL
    # ================================================================================
elif name == 'N08-E-A-DTSL':
    divided_file.loc[:, 'Participant name'] = name[0:12]
    name = 'N08-E-A-DTSL-LT-DE'

elif name == 'N08-E-A-DTSL 2 ':
    print('FILE DELETED: ', name)
    tsv_flag = False
    continue

elif name == 'N08-E-A-DTSL 3':
    divided_file.loc[:, 'Participant name'] = name[0:12]
    name = 'N08-E-A-DTSL-LF-LA'

elif name == 'N08-E-A-DTSL-E':
    print('FILE DELETED: ', name)
    tsv_flag = False
    continue
```

```
            # ================================================================================
            # N08-E-A-DTSL
            # ================================================================================
            # ================================================================================
            # N12-E-B-TDSL
            # ================================================================================
        elif name == 'N12-E-B-TDSL':
            divided_file.loc[:, 'Participant name'] = name[0:12]
            name = 'N12-E-B-TDSL-LT-DE'

        elif name == 'N12-E-B-TDSL - 2 ':
            divided_file.loc[:, 'Participant name'] = name[0:12]
            name = 'N12-E-B-TDSL-LF-LA'
            # ================================================================================
            # N12-E-B-TDSL
            # ================================================================================
            # ================================================================================
            # N12-E-B-TDSL
            # ================================================================================
        elif name == 'N13-E-B-LDST':
            divided_file.loc[:, 'Participant name'] = name[0:12]
            name = 'N13-E-B-LDST-LF-LA'

        elif name == 'N13-E-B-LDST - TURN - DECEND':
            divided_file.loc[:, 'Participant name'] = name[0:12]
            name = 'N13-E-B-LDST-LT-DE'
            # ================================================================================
            # N12-E-B-TDSL
            # ================================================================================

        elif len(name) > 12:
            print('MORE THAN 12 CHAR.')
            print(name[0:12])

            divided_file.loc[:, 'Participant name'] = name[0:12]

        if tsv_flag == True:
            divided_file.to_csv(name + '.tsv', sep='\t', encoding = 'utf-16', index=False, header=True)
            print('**************************************************')

else:
```

```
    print('=============================================================================================')
    print('=============================================================================================')
    print('=============================================================================================')
    print('                                    NOT DIVIDING ANY FILE                                    ')
    print('=============================================================================================')
    print('=============================================================================================')
    print('=============================================================================================')

        ##############################################################################################################
        ##############################################################################################################
        ##############################################################################################################
        ########################################################################## FOR DIVIDING MASTER FILES TO PARTICIPANTS FILES #######
        ##############################################################################################################
        ##############################################################################################################
        ##############################################################################################################

        ##############################################################################################################
        ##############################################################################################################
        ##############################################################################################################
        ########################################################################## CREATING ONE EMPTY DATAFRAME TO KEEP ALL DATA #########
        ##############################################################################################################
        ##############################################################################################################
        ##############################################################################################################
master_dataframe = pd.DataFrame()
master_level_flight = pd.DataFrame()
master_level_turn = pd.DataFrame()
master_descent = pd.DataFrame()
master_landing = pd.DataFrame()
synchronized_data = pd.DataFrame()
        ##############################################################################################################
        ##############################################################################################################
        ##############################################################################################################
        ########################################################################## CREATING ONE EMPTY DATAFRAME TO KEEP ALL DATA #########
        ##############################################################################################################
        ##############################################################################################################
        ##############################################################################################################

        ##############################################################################################################
        ##############################################################################################################
        ##############################################################################################################
        ########################################################################## READING DATA POINTS OF RUNWAY ##########################
```

```
        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################

# surface of runway
cloud_points_runway = pd.read_excel('runway_cloud.xlsx')
cloud_points_runway = cloud_points_runway.drop_duplicates(subset=['____X,____m', '____Z,____m'], keep = 'first')

        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################
        ############################################################################ READING THE RESEARCH FILES'S NAME ############
        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################

importing_tsv_files =  ['N10-C-B-LTDS']
#
#importing_tsv_files =  [
#                         'E01-E-B-LSDT',                                    'E01-E-A-STDL',
#
#                         'N01-C-B-TSDL',                                    'N01-C-A-LTSD',
#                         'N02-C-B-TSDL',                                    'N02-C-A-DSTL',
#                         'N03-C-B-TSDL',                                    'N03-C-A-DSTL','N03-C-A-DSTL-LF',
#                         'N04-C-A-STDL',                                    'N04-C-B-LDST',
#                         'N05-C-B-LSTD',                                    'N05-C-A-STDL','N05-C-A-STDL-LT',
#                         'N06-C-B-LTDS',                                    'N06-C-A-LDTS',
#                         'N07-C-B-SDTL',                                    'N07-C-A-DSTL',
#                         'N08-C-B-LDTS',                                    'N08-C-A-LSTD',
#                         'N09-C-B-LTDS',                                    'N09-C-A-STDL',
#                         'N10-C-B-LTDS',                                    'N10-C-A-STDL',
#
#                         'N07-E-B-LSTD',                                    'N07-E-A-LSTD-DE','N07-E-A-LSTD-LA-LF', 'N07-E-A-LSTD-
#                         'N08-E-B-LDST',                                    'N08-E-A-DTSL-LF-LA', 'N08-E-A-DTSL-LT-DE',
#                         'N09-E-B-LTDS',                                    'N09-E-A-SDTL',
#                         'N10-E-B-DTSL',                                    'N10-E-A-LTDS',
#                         'N11-E-B-DSTL',                                    'N11-E-A-LSDT',
#                         'N12-E-B-TDSL-LF-LA','N12-E-B-TDSL-LT-DE',         'N12-E-A-LTSD',
#                         'N13-E-B-LDST-LF-LA','N13-E-B-LDST-LT-DE',         'N13-E-A-LTDS',
#                         'N14-E-B-LDST',                                    'N14-E-A-SDTL',
#                         'N15-E-B-LSTD',                                    'N15-E-A-LDST'
```

```
#                         ]

p = 0

for file_name in importing_tsv_files:
    p += 1
    print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    print('FILENAME: ', file_name, '               ',p, ' OUT OF:', len(importing_tsv_files),                              'START_
    print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    print('')

    print('************************************************************************************************************************
    print('************************************************************************************************************************
    print('************************************************************************************************************************
    print('************************************************************************************************************************
    print(importing_tsv_files)
    print('************************************************************************************************************************
    print('************************************************************************************************************************
    print('************************************************************************************************************************
    print('************************************************************************************************************************
    print('')
#   ************************************************************************
    time.sleep(5)

    if file_name == 'N03-C-A-DSTL-LF' or file_name == 'N05-C-A-STDL-LT' or file_name == 'N07-E-A-LSTD-DE':
        filename_eye = file_name + '.tsv'           # [:12]
        filename_flight = file_name + '.csv'    # [:12]
        print(filename_eye, '==> ', filename_flight)

    else:
        print('**********************************')
        filename_eye = file_name + '.tsv'           # [:12]
        filename_flight = file_name[:12] + '.csv'
        print(filename_eye, '==> ', filename_flight)
        print('**********************************')

    print('************************************************************************************************************************
```

```
    print('READ THE .TSV FILE PARSING THE DATE DATA')
    print('*************************************************************************************************************')
    print('')

    df_eye = pd.read_csv(filename_eye, encoding = 'utf-16', delimiter='\t', low_memory=False, parse_dates = {'time':['Recording dat
    df_eye['Recording timestamp'] = pd.to_timedelta(df_eye['Recording timestamp'], unit ='ms')
    df_eye['time'] = df_eye['time'] + df_eye['Recording timestamp']

    df_eye['scenario duration'] = np.nan
    df_eye['scenario duration'] = pd.to_timedelta(df_eye['scenario duration'])

    list_extra = []
    list_extra = df_eye['Event'][df_eye['Event'].notnull()].unique()
    list_extra = list_extra.tolist()
    list_extra.extend(['X', 'XX'])

    list_events = ['Landing', 'Level Turn', 'Level Flight', 'Descent']

#     print('********************** The Tobii EVENTS **********************')
#     print('********************** The Tobii EVENTS **********************')
#     print('********************** The Tobii EVENTS **********************')
#     print(list_extra)
#     print('********************** The Tobii EVENTS **********************')
#     print('********************** The Tobii EVENTS **********************')
#     print('********************** The Tobii EVENTS **********************')
#     print('')
#     print('>>>>>>>>>>>>>>>>>>>>>>> Desired EVENTS <<<<<<<<<<<<<<<<<<<<<<<')
#     print('>>>>>>>>>>>>>>>>>>>>>>> Desired EVENTS <<<<<<<<<<<<<<<<<<<<<<<')
#     print('>>>>>>>>>>>>>>>>>>>>>>> Desired EVENTS <<<<<<<<<<<<<<<<<<<<<<<')
#     print(list_events)
#     print('>>>>>>>>>>>>>>>>>>>>>>> Desired EVENTS <<<<<<<<<<<<<<<<<<<<<<<')
#     print('>>>>>>>>>>>>>>>>>>>>>>> Desired EVENTS <<<<<<<<<<<<<<<<<<<<<<<')
#     print('>>>>>>>>>>>>>>>>>>>>>>> Desired EVENTS <<<<<<<<<<<<<<<<<<<<<<<')
    print('')
    for i in list_events:
        if i in list_extra:
            list_extra.remove(i)

#     print('++++++++++++++++++++++ Unwanted EVENTS ++++++++++++++++++++++')
#     print('++++++++++++++++++++++ Unwanted EVENTS ++++++++++++++++++++++')
#     print('++++++++++++++++++++++ Unwanted EVENTS ++++++++++++++++++++++')
```

```
#     print(list_extra)
#     print('++++++++++++++++++++++ Unwanted EVENTS ++++++++++++++++++++++')
#     print('++++++++++++++++++++++ Unwanted EVENTS ++++++++++++++++++++++')
#     print('++++++++++++++++++++++ Unwanted EVENTS ++++++++++++++++++++++')
#     print('')
    for i in list_extra:
        df_eye['Event'] = df_eye['Event'].str.replace(i, '')

    df_eye['Event'] = df_eye['Event'].fillna('')
    df_eye['Event'] = df_eye['Event'].replace({'': np.NaN})

    print('======================== The KEPT EVENTS ========================')
    print('======================== The KEPT EVENTS ========================')
    print('======================== The KEPT EVENTS ========================')
    print(df_eye['Event'][df_eye['Event'].notnull()].unique())
    print('======================== The KEPT EVENTS ========================')
    print('======================== The KEPT EVENTS ========================')
    print('======================== The KEPT EVENTS ========================')
    print('')
    df_eye = df_eye[~((df_eye['Recording timestamp'].duplicated(keep=False)) & (df_eye['Event'].isnull()))].reset_index(drop=True)

    df_eye['Event'].astype(str)
    flag_event = False

    zero_point = 0

        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################
        ############################################################################ GENERATING TAGS FOR EVENTS / SCENARIOS / COMPUTING TIME
        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################

    print('GENERATING TAGS FOR EVENTS / SCENARIOS')
    print('')
    for index, row in df_eye.iterrows():
        if  len(df_eye['Event']) == index + 1:
            break

        elif pd.isnull(df_eye['Event'].iloc[index]) == True and pd.isnull(df_eye['Event'].iloc[index+1]) == True:
```

```
        flag_event = False

    elif pd.isnull(df_eye['Event'].iloc[index]) == False and pd.isnull(df_eye['Event'].iloc[index+1]) == True and flag_event ==

        if index == 0:
            if pd.isnull(df_eye['Event'].iloc[index]) == False and pd.isnull(df_eye['Event'].iloc[index+1]) == True:
                zero_point = index
        else:

            if pd.isnull(df_eye['Event'].iloc[index]) == False and pd.isnull(df_eye['Event'].iloc[index+1]) == True and pd.isnu
                zero_point = index

        df_eye.at[index+1, 'Event'] = df_eye['Event'].iloc[index]

        df_eye.at[index, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index] - df_eye['Recording timestamp'].iloc[

        flag_event = False

    elif pd.isnull(df_eye['Event'].iloc[index]) == False and pd.isnull(df_eye['Event'].iloc[index+1]) == False:

        df_eye.at[index, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index]
        df_eye.at[index+1, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index+1]
        df_eye.at[index, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index] - df_eye['Recording timestamp'].iloc[
        df_eye.at[index+1, 'scenario duration'] = df_eye['Recording timestamp'].iloc[index+1] - df_eye['Recording timestamp'].i

        zero_point = 0
        flag_event = True
    ##########################################################################################################################
    ##########################################################################################################################
    ##########################################################################################################################
    ############################################################################ ORDERING COLUMNS OF EYE TRACKERING DATA ##############
    ##########################################################################################################################
    ##########################################################################################################################
    ##########################################################################################################################

print('ORDING THE COLUMNS AND GENERATING TAGS FOR PARTICIPANTS / GROUPS / TRAINING')
print('')
df_eye['Participant name'] = df_eye['Participant name'].str[:12]
df_eye['Number'] = df_eye['Participant name'].str[1:3]
df_eye['Group'] = np.where(df_eye['Participant name'].str[4:5]=='E', 'Experimental', 'Control')
df_eye['Skill'] = np.where(df_eye['Participant name'].str[0:1]=='E', 'Expert', 'Novice')
```

```
df_eye['Training'] = np.where(df_eye['Participant name'].str[6:7]=='A', 'After Treatment', 'Before Treatment')

# if this is fixation or not ( 1 is yes and 0 is no)
df_eye['fixation'] = 0

df_eye = df_eye[['time', 'Recording timestamp', 'scenario duration', 'Participant name', 'Skill' , 'Group', 'Number', 'Training
                 'Gaze event duration', 'Pupil diameter left', 'Pupil diameter right',
              'AOI hit [XXX - AI]', 'AOI hit [XXX - ALT]', 'AOI hit [XXX - ASI]',
       'AOI hit [XXX - CONT]', 'AOI hit [XXX - GAG]',
       'AOI hit [XXX - HI]', 'AOI hit [XXX - OSW]',
       'AOI hit [XXX - RT]', 'AOI hit [XXX - TAC]', 'AOI hit [XXX - TC]',
       'AOI hit [XXX - VSI]']]

#df_interpolated['AOI'] = df_interpolated.apply(func=lambda row: AOI(row, 14, 22), axis=1)

#df_eye = df_eye.set_index(['time'])

df_eye['sum'] = df_eye.iloc[:, df_eye.columns.get_loc('AOI hit [XXX - AI]'):df_eye.columns.get_loc('AOI hit [XXX - VSI]')+1].ap

df_eye = df_eye[['time','Recording timestamp', 'scenario duration', 'Participant name', 'Skill', 'Group', 'Number', 'Training',
                 'Gaze event duration', 'Pupil diameter left', 'Pupil diameter right',
              'AOI hit [XXX - AI]', 'AOI hit [XXX - ALT]', 'AOI hit [XXX - ASI]',
       'AOI hit [XXX - CONT]', 'AOI hit [XXX - GAG]',
       'AOI hit [XXX - HI]', 'AOI hit [XXX - OSW]',
       'AOI hit [XXX - RT]', 'AOI hit [XXX - TAC]', 'AOI hit [XXX - TC]',
       'AOI hit [XXX - VSI]']]


df_eye = df_eye.dropna(subset=['Event'])

# keeping the desired columns with desired order
df_eye = df_eye[['time','Recording timestamp', 'scenario duration', 'Participant name', 'Skill', 'Group', 'Number', 'Training',
                          'AOI hit [XXX - AI]', 'AOI hit [XXX - ALT]', 'AOI hit [XXX - ASI]',
       'AOI hit [XXX - CONT]', 'AOI hit [XXX - GAG]',
       'AOI hit [XXX - HI]', 'AOI hit [XXX - OSW]',
       'AOI hit [XXX - RT]', 'AOI hit [XXX - TAC]', 'AOI hit [XXX - TC]',
       'AOI hit [XXX - VSI]']]

df_eye.columns = ['time', 'timestamp', 'scenario time', 'Participant name', 'Skill', 'Group', 'Number', 'Training', 'Event', 'E
                        'AI', 'ALT', 'ASI',
                        'CONT', 'GAG',
```

```
                        'HI', 'OSW',
                        'RT', 'TAC', 'TC',
                        'VSI']

    AOI_list =    ['AI',           'ALT',           'HI',          'TC',         'ASI' ,        'VSI',              'TAC',

    # finding the column (AOI) that partcipants looked at
    df_aoi = df_eye.iloc[:, df_eye.columns.get_loc('AI'):df_eye.columns.get_loc('VSI')+1].idxmax(axis=1)

    # adding the AOI to the dataframe / AOI is the visited Area of Interst
    df_eye['AOI'] =  df_aoi

    # if the eye movement is not fixation, then AOI is 0
    df_eye['AOI'] = df_eye.apply(lambda x: np.nan if x['Eye movement type'] != 'Fixation' else x['AOI'], axis=1)
    df_eye['AOI'] = df_eye.apply(lambda x: np.nan if x['Eye movement type'] == 'Fixation' and x['sum'] == 0 else x['AOI'], axis=1)

    #print('++++++++++++++++++++++++++++++++++++++++')
    #print(df_eye_data.iloc[72460:72470, : ])

    if file_name == 'N08-C-B-LDTS':
        df_eye['AOI'].loc[df_eye['timestamp'] == '00:04:12.993000000'] = 'CONT'

    print('DIVIDING THE DATAFRAME BEFORE THE LOOP')
    print('')



#    df_eye_landing = df_eye.loc[df_eye['Event'] == 'Landing'].reset_index(drop=True)
#    df_eye_levelturn = df_eye.loc[df_eye['Event'] == 'Level Turn'].reset_index(drop=True)
#    df_eye_levelflight = df_eye.loc[df_eye['Event'] == 'Level Flight'].reset_index(drop=True)
#    df_eye_descent = df_eye.loc[df_eye['Event'] == 'Descent'].reset_index(drop=True)

    # I normalized the experiment time and make it equal for all participants


    df_eye_landing = df_eye.loc[df_eye['Event'] == 'Landing'].reset_index(drop=True)
    df_eye_levelturn = df_eye.loc[df_eye['Event'] == 'Level Turn'].reset_index(drop=True)
    df_eye_levelflight = df_eye.loc[df_eye['Event'] == 'Level Flight'].loc[df_eye['scenario time'] < '00:01:00'].reset_index(drop=T
    df_eye_descent = df_eye.loc[df_eye['Event'] == 'Descent'].reset_index(drop=True)

#    df_eye.loc[df_eye['Event'] == 'Landing'].loc[df_eye['scenario time'] < '00:01:00']
```

```
df_eye_landing.name = 'df_eye_landing'
df_eye_levelturn.name = 'df_eye_levelturn'
df_eye_levelflight.name = 'df_eye_levelflight'
df_eye_descent.name = 'df_eye_descent'

    ##########################################################################################################################
    ##########################################################################################################################
    ##########################################################################################################################
    ############################################################## GENERATING TAGS FOR EVENTS / SCENARIOS ###################
    ##########################################################################################################################
    ##########################################################################################################################
    ##########################################################################################################################

    ##########################################################################################################################
    ##########################################################################################################################
    ##########################################################################################################################
    ############################################################## REGRESSION FOR AIRSPEED ###################################
    ##########################################################################################################################
    ##########################################################################################################################
    ##########################################################################################################################

print('REGRESSION ==> BEFORE ENTERING THE LOOP')
print('')
list_data=[]

for filename in ['Data_20_40.xlsx' , 'Data_15_35.xlsx']:
     data = pd.read_excel(filename)
     list_data.append(data)

dataset = pd.concat(list_data, ignore_index=True)

dataset = dataset[['Vtrue,_ktas', '_Vind,_kias', '__alt,__ind', 'pitch,__deg', '_roll,__deg', 'hding,_true', 'hding,__mag', 'al

X = dataset[['_Vind,_kias', '__alt,__ind']]
y = dataset['Vtrue,_ktas']

poly_features= PolynomialFeatures(degree = 2)
X_poly = poly_features.fit_transform(X)
poly_features.get_feature_names()
```

```
    model = Pipeline([('poly', PolynomialFeatures(degree=2)), ('linear', LinearRegression(fit_intercept=False))])
    model = model.fit(X, y)

        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################
        ########################################################################## REGRESSION FOR AIRSPEED ########################
        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################

        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################
        ########################################################################## THE RESEARCH SCENARIO  ########################
        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################
    scenario_list = [df_eye_landing]
#     scenario_list = [df_eye_landing]
        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################
        ########################################################################## THE RESEARCH SCENARIO  ########################
        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################

    for scenario in scenario_list:
        if scenario.empty:
            print('================================================================================')
            print('================================================================================')
            print('================================================================================')
            print('================================================================================')
            print('================================================================================')
            print('                                EMPTY DATAFRAME: ', scenario.name, '====> ', file_name)
            print('================================================================================')
            print('================================================================================')
            print('================================================================================')
            print('================================================================================')
```

```
        print('================================================================================')
        print('')
        continue

    else:
        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
        print('                                                    NOT EMPTY DATAFRAME', scenario.name)
        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
        print('')

        if scenario.name == 'df_eye_landing':
            task_name = 'Landing'

        elif scenario.name == 'df_eye_levelturn':
            task_name = 'Level_Turn'

        elif scenario.name == 'df_eye_levelflight':
            task_name = 'Level_Flight'

        else:
            task_name = 'Descent'

        ##################################################################################################################################
        ##################################################################################################################################
        ##################################################################################################################################
        ############################################################## REGRESSION FOR AIRSPEED ###########################################
        ##################################################################################################################################
        ##################################################################################################################################
        ##################################################################################################################################

        ##################################################################################################################################
        ##################################################################################################################################
        ##################################################################################################################################
```

```
######################################################################## DEFINING VARIABLES AND DEFINING FIXATION COUNTS AND
#####################################################################################################################################
#####################################################################################################################################
#####################################################################################################################################
#
fixation_count = 0
fixation_duration = 0
# defining a variable as counter of rows of dataframe
row_index = 0

columns = ['Index','timestamp', 'scenario time', 'AOI', 'Duration']
index = range(0,0)
df_fixation = pd.DataFrame(index = index, columns = columns)

print('################### S T A R T ################### Number of records: ' , len(scenario), '---: ', scenario_list[0
print('')
# It counts number of fixation, fixation duration
for index, row in scenario.iterrows():
    counter_order = row_index
    # fixing the AOI, moving forward , I have to make sure it not exceeding the

    if row_index == len(scenario) - 1:
        break

    while (index <= len(scenario) - 1 and counter_order <= len(scenario) - 1  and scenario['Eye movement type'].iloc[ro
           scenario['Eye movement type'].iloc[counter_order] == 'Fixation' and
           scenario['sum'].iloc[counter_order]  == 0 and
           scenario['sum'].iloc[row_index - 1 ] == 1 and
           scenario['Gaze event duration'].iloc[row_index - 1] == scenario['Gaze event duration'].iloc[counter_order]):

        scenario['AOI'].iloc[counter_order] = scenario['AOI'].iloc[row_index -1]
        scenario['sum'].iloc[counter_order] = 1
        scenario[scenario['AOI'].iloc[row_index -1]].iloc[counter_order] = scenario[scenario['AOI'].iloc[row_index -1]]
        counter_order = counter_order + 1

    counter_reverse = row_index

    while (index <= len(scenario) - 1 and counter_reverse != 0 and scenario['Eye movement type'].iloc[row_index + 1] ==
           scenario['Eye movement type'].iloc[counter_reverse ] == 'Fixation' and
           scenario['sum'].iloc[counter_reverse]  == 0 and
           scenario['sum'].iloc[row_index + 1] == 1 and
```

```
            scenario['Gaze event duration'].iloc[row_index + 1] == scenario['Gaze event duration'].iloc[row_index]):

            scenario['AOI'].iloc[counter_reverse] = scenario['AOI'].iloc[row_index + 1]
            scenario['sum'].iloc[counter_reverse] = 1
            scenario[scenario['AOI'].iloc[row_index +1]].iloc[counter_reverse] = scenario[scenario['AOI'].iloc[row_index +1

            counter_reverse = counter_reverse - 1

        # 1st check: that it is not the last row of dataframe, when you get to one row to the last row, do a few tasks and
        # before that it says that if the cell (current row) is not fixation but the next one is, then it is a fixation
        row_index += 1

print('======================================== FIXATION DURAION ========================================')
print('')
row_index = 0
for index, row in scenario.iterrows():

    if row_index == len(scenario) - 1:
        break

    if len(scenario['Event'])-1 == row_index + 1 :
        if  scenario['Eye movement type'].iloc[row_index] != 'Fixation' and scenario['Eye movement type'].iloc[row_inde
            fixation_count = fixation_count + 1
            scenario.at[row_index+1, 'fixation'] = 1

            fixation_duration = fixation_duration + scenario['Gaze event duration'].iloc[row_index+1]

        # counter_fixation + 1 is the index of fixation not counter_fixation
        # it addes the index of fixation, AOI, and gaze duration to a series, then it appened this to the dataframe
        #++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        s = pd.Series([row_index+1, scenario['timestamp'].iloc[row_index+1], scenario['scenario time'].iloc[row_index+1
                       scenario['Gaze event duration'].iloc[row_index+1]], index=['Index', 'timestamp', 'scenario t
        df_fixation = df_fixation.append(s, ignore_index=True)

        break

    #2nd check:if this is the first row and eye movement is fixation, it is a fixation
    elif row_index == 0 and scenario['Eye movement type'].iloc[row_index] == 'Fixation':


        fixation_count = fixation_count + 1
```

```
            scenario.at[row_index, 'fixation'] = 1
            fixation_duration = fixation_duration + scenario['Gaze event duration'].iloc[row_index+1]

            s = pd.Series([row_index+1, scenario['timestamp'].iloc[row_index+1], scenario['scenario time'].iloc[row_index+1
                           scenario['Gaze event duration'].iloc[row_index+1]], index=['Index', 'timestamp', 'scenario t
            df_fixation = df_fixation.append(s, ignore_index=True)

        #3rd check:if this is not the first and last row, and this is not fixaion but then next one is, then it is a fixati
        elif scenario['Eye movement type'].iloc[row_index] != 'Fixation' and scenario['Eye movement type'].iloc[row_index+1

            #print('++++++++++++++++++++++++++++++++++++++++', fixation_count)
            #print(s)

            #print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~', fixation_count)
            fixation_count = fixation_count + 1
            scenario.at[row_index+1, 'fixation'] = 1
            fixation_duration = fixation_duration + scenario['Gaze event duration'].iloc[row_index+1]

            s = pd.Series([row_index+1, scenario['timestamp'].iloc[row_index+1], scenario['scenario time'].iloc[row_index+1
                        index=['Index', 'timestamp', 'scenario time', 'AOI', 'Duration'])
            df_fixation = df_fixation.append(s, ignore_index=True)

        # it addes one to row counter
        row_index += 1

    #print('number of fixation', fixation_count)
    #print('fixation duration', fixation_duration)
    #print('ave fixatin duration', fixation_duration/fixation_count)

    df_fixation.to_csv('_'  + file_name + '_' + task_name + '_' + 'checking_AOIs.csv', encoding='utf-8')

    #print('++++++++++++++++++++++++++++++++++++++')

    ##########################################################################################################################
    ##########################################################################################################################
    ##########################################################################################################################
    ############################################################## DEFINING VARIABLES AND DEFINING FIXATION COUNTS AND
    ##########################################################################################################################
    ##########################################################################################################################
    ##########################################################################################################################
```

```
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
############################################################################ TRANSITION MATRIX ############################
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################

print('TRANSITION MATRIX')
columns = ['From','To']
index = range(0, math.ceil(len(df_fixation)/2))

df_from_to = pd.DataFrame(index = index, columns = columns)

counter_index = 0

for i in range(0, math.ceil(len(df_fixation)/2)):
    #time.sleep(0.01)
    #print(counter_index, i, df_fixation['AOI'].iloc[counter_index], df_fixation['AOI'].iloc[counter_index+1])
    df_from_to['From'].iloc[i] = df_fixation['AOI'].iloc[counter_index]
    df_from_to['To'].iloc[i] = df_fixation['AOI'].iloc[counter_index+1]

    counter_index += 1

# INDENT
df_from_to = df_from_to[df_from_to['From'] != df_from_to['To']]
df_from_to = df_from_to.reset_index(drop=True)
df_crosstab = pd.crosstab(df_from_to.From, df_from_to.To)

#scenario.to_csv('df_crosstab.csv', encoding='utf-8')
#df_fxation.loc[scenario['Eye movement type'] == 'Fixation']
#df_eye_levelturn[['timestamp','Eye movement type', 'fixation', 'AOI']].loc[df_eye_levelturn['AOI'].isnull() = True]


#df_eye_levelturn.to_csv('df_eye_levelturn.csv', encoding='utf-8')

print('checking the data before making transition matrix') #, df_eye_levelturn[['timestamp','Eye movement type', 'fixat
#df_eye_levelturn[['timestamp','Eye movement type', 'fixation', 'AOI']][df_eye_levelturn['AOI'].isnull()].loc[df_eye_le


for i in AOI_list:
```

```
            if (df_from_to['From'].str.contains(i).any() == False and df_from_to['To'].str.contains(i).any() == False) or \
            (df_from_to['From'].str.contains(i).any() == True and df_from_to['To'].str.contains(i).any() == False) or \
            (df_from_to['From'].str.contains(i).any() == False and df_from_to['To'].str.contains(i).any() == True):

                s = pd.Series([i, i], index=['From', 'To'])
                df_from_to = df_from_to.append(s, ignore_index=True)

        df_crosstab = pd.crosstab(df_from_to.From, df_from_to.To)

        for i in AOI_list:
            df_crosstab[i][i] = 0

#         print('++++++++++++++++++++++++++++++++++++++++++++++++++')
#         print(df_crosstab)
#         print('++++++++++++++++++++++++++++++++++++++++++++++++++')

        # heat map of transion matrix

        df_crosstab_plot = df_crosstab

        df_from_to['numeric_from'] = pd.Categorical(df_from_to.From)
        df_from_to['numeric_from'] = df_from_to['numeric_from'].cat.codes
        df_from_to['numeric_to'] = pd.Categorical(df_from_to.To)
        df_from_to['numeric_to'] = df_from_to['numeric_to'].cat.codes

        #transition_matrix = pd.DataFrame(df_crosstab)
        #transition_matrix

        movements_from= list(df_from_to['numeric_from'])
        movements_to= list(df_from_to['numeric_to'])
        movements_from.append(movements_to[-1])

        transition_matrix_list = [[0]*len(df_crosstab) for _ in range(len(df_crosstab))]

        for i in range(0, len(movements_from)):
            if i+1 < len(movements_from):
                transition_matrix_list[movements_from[i]][movements_from[i+1]] += 1

        transition_matrix = pd.DataFrame(transition_matrix_list)

        for i in range(len(AOI_list)):
```

```
    transition_matrix[i][i]=0

transition_matrix.to_csv('_'  + file_name + '_' + task_name + '_' + 'Transition_Matrix.csv', encoding='utf-8')

#########################################################################################################################
#########################################################################################################################
#########################################################################################################################
############################################################################ TRANSITION MATRIX ############################
#########################################################################################################################
#########################################################################################################################
#########################################################################################################################


#########################################################################################################################
#########################################################################################################################
#########################################################################################################################
############################################################################ VISUAL ENTROPY ###############################
#########################################################################################################################
#########################################################################################################################
#########################################################################################################################

probability_matrix = transition_matrix.div(transition_matrix.values.sum())
log_probability_matrix = probability_matrix.copy()
log_probability_matrix.iloc[:,:] = 0

for i in range (0, len(AOI_list)):
    log_probability_matrix[i] = probability_matrix[i].apply(lambda x: math.log2(x) if x > 0 else 0)

entropy_matrix = probability_matrix * log_probability_matrix
entropy = -entropy_matrix.values.sum()


scenario['entropy'] = entropy
print('ENTROPY: ', entropy)

df_crosstab.to_csv('_'  + file_name + '_' + task_name + '_' + 'CrossTab.csv', encoding='utf-8')

#########################################################################################################################
#########################################################################################################################
#########################################################################################################################
                                                                                    # PLOTTING TRANSITION MATRIX
```

```
#########################################################################################################################
#########################################################################################################################
#########################################################################################################################

style.use('classic')

# heatmap of transion matrix
# cmap='Set3'

f, ax = plt.subplots(figsize=(8, 8))


#my_cmap ='Set3'
my_cmap = sns.light_palette("Navy", as_cmap=True)

sns.set(font_scale = 1.0)
sns.heatmap(df_crosstab_plot, annot=True, cbar=False, cmap=my_cmap, fmt='d', square=True, linewidths= 1.0, annot_kws={"
ax.figure.axes[0].yaxis.label.set_size(6)

a1 = ax.get_xlabel()
a2 = ax.get_ylabel()

ax.set_xlabel(a1, fontsize = 14)
ax.set_ylabel(a2, fontsize = 14)
#ax.get_xaxis().set_label_coords(0,0.5)
ax.get_yaxis().set_label_coords(-0.05, 0.5)
ax.get_xaxis().set_label_coords(0.5, -0.05)
ax.set_title('Visual Entropy = '+ str(np.round_(entropy,2)), loc='left', fontname='arial', fontsize=10)

plt.savefig('_' + file_name + '_' + task_name + '_'+'Transition_Matrix.png')
plt.savefig('_' + file_name + '_' + task_name + '_'+'Transition_Matrix.pdf')

#########################################################################################################################
#########################################################################################################################
#########################################################################################################################
                                                                        # PLOTTING TRANSITION MATRIX
#########################################################################################################################
#########################################################################################################################
#########################################################################################################################

df_crosstab_transposed = df_crosstab.T
```

```
df_crosstab_transposed_unstacked = df_crosstab_transposed.unstack()

##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
############################################################################ TRANSITION MATRIX ANALYSIS USING SPSS #############
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################

df_spss = df_crosstab_transposed_unstacked.reset_index()
df_spss.columns = ['from', 'to', 'count']

df_spss['zero'] = 1

map_AOI = {'AI': 1, 'ALT': 2, 'ASI': 3, 'CONT': 4, 'GAG':5, 'HI':6, 'OSW':7, 'RT':8, 'TAC':9, 'TC':10, 'VSI':11}

df_spss['from'] = df_spss['from'].map(map_AOI)
df_spss['to'] = df_spss['to'].map(map_AOI)
df_spss['zero'] = df_spss.apply(lambda x: 0 if x['from'] == x['to'] else 1, axis=1)
df_spss.to_csv('_'  + file_name + '_' + task_name + '_' + 'SPSS.csv', encoding='utf-8')

scenario[['Gaze point X', 'Gaze point Y']] = scenario[['Gaze point X', 'Gaze point Y']].interpolate(method='linear', li

scenario = scenario.reset_index()

#   [df_eye_landing, df_eye_levelturn, df_eye_levelflight, df_eye_descent]
#   for scenario in [df_eye_levelturn]: # , df_eye_levelturn, df_eye_levelflight, df_eye_descent

#   this part should be added to dataframe of eye
rng = pd.date_range(scenario['time'].dt.round('100ms').iloc[0], periods=(scenario['time'].iloc[-1] - scenario['time'].i

#time.sleep(5)
eye_to_append = pd.DataFrame(rng)
eye_to_append.columns = ['time']

# ********************************************************************************************
# creating a new dataframe with the stamp of 0
# ********************************************************************************************
# create a list of df_eye's columns
```

```
list_columns = [x for x in scenario.columns]
dataframe_eye = scenario[list_columns].copy()
dataframe_eye['stamped'] = 0

for col in dataframe_eye.columns[:]:
    if col != 'time':
        eye_to_append[col] = np.nan

# RW: Mark the new (interpolated) data as "stamped"
eye_to_append['stamped'] = 1
print('5')
eye_data = dataframe_eye.append(eye_to_append, ignore_index = True).sort_values('time').reset_index(drop=True) #.set_in
print('6')
eye_data[['Participant name', 'Skill', 'Group', 'Number', 'Training', 'Event', 'Eye movement type', 'Gaze event duratio
print('7')
#    eye_data[['Gaze point X', 'Gaze point Y']] = eye_data[['Gaze point X', 'Gaze point Y']].interpolate(method='linear
#    print(eye_data.head(5))
#    eye_data.interpolate(limit_direction='both', inplace = True)
#    print (eye_data.iloc[0:5, 0:10])
#    print (eye_data.iloc[0:5, 10:20])
#    eye_data[['Gaze point X', 'Gaze point Y']]
#    creating tag 2 for fixation of those point transfered to stamped 1 points
#    tag 1 is for real fixation point , 2 for transfered one

row_index = 0
for index, row in eye_data.iterrows():
    #print(index)
    counter_reverse = row_index

    while (counter_reverse > 0 and eye_data['Eye movement type'].iloc[row_index] == 'Fixation' and
           eye_data['Eye movement type'].iloc[counter_reverse - 1 ] == 'Fixation' and
           eye_data['stamped'].iloc[row_index] == 1 and
           pd.isnull(eye_data['fixation'].iloc[row_index]) == True):

        if eye_data['fixation'].iloc[counter_reverse - 1] == 2 and eye_data['stamped'].iloc[counter_reverse - 1] == 1:
            break

        elif eye_data['fixation'].iloc[counter_reverse - 1] == 1 and eye_data['stamped'].iloc[counter_reverse - 1] == 0
            eye_data.at[row_index, 'fixation'] = 2
            #print('TAG')
```

```
        counter_reverse = counter_reverse - 1

    if pd.isnull(row['timestamp']) == True:

        eye_data.at[index, 'timestamp'] = (eye_data['time'].iloc[index] - eye_data['time'].iloc[index-1]) + eye_data['t
        eye_data.at[index, 'scenario time'] = (eye_data['time'].iloc[index] - eye_data['time'].iloc[index-1]) + eye_dat

    row_index += 1

    eye_data['fixation'] = eye_data['fixation'].replace('NaN', 0)

# eye_data[['Gaze point X', 'Gaze point Y']]

eye_data = eye_data.set_index('time')
eye_data[['Gaze point X', 'Gaze point Y']] = eye_data[['Gaze point X', 'Gaze point Y']].interpolate(method='linear', li
eye_data = eye_data.reset_index()


#eye_data[['Gaze point X', 'Gaze point Y']] = eye_data[['Gaze point X', 'Gaze point Y']].interpolate(method='linear', l
#eye_data['fixation'] = eye_data['fixation'].fillna(0)

if eye_data['stamped'].iloc[0] == 1:
    eye_data = eye_data.drop(eye_data.index[0]).reset_index(drop=True)

# eye_data[['Gaze point X', 'Gaze point Y']]

# RW: This is the data you want
eye_data[['Gaze point X', 'Gaze point Y']] = eye_data[['Gaze point X', 'Gaze point Y']].fillna(method='bfill')

eye_final = eye_data.query('stamped == 1').drop('stamped', 1).reset_index(drop=True)
eye_final = eye_final.set_index('time')

                                    # THE END OF CODING FOR EYE MOVEMENTS
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

#eye_data['fixation'].isnull().values.any()

eye_data['fixation'] = eye_data['fixation'].fillna(0)

#eye_final[['Gaze point X', 'Gaze point Y']].isnull().sum(axis = 0)
#eye_final[['Gaze point X', 'Gaze point Y']]
```

```
eye_final[['Gaze point X', 'Gaze point Y']] = eye_final[['Gaze point X', 'Gaze point Y']].astype(int)

##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
############################################################################ EYE MOVEMENTS CLUSTERING ANALYSIS ############
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################

print('++++++++++++++++++++++++++++++++++++++++++++++++ CLSUTERING ++++++++++++++++++++++++++++++++++++++++++++++')

clustering_part = 0
if clustering_part == 1:
    print('++++++++++++++++++++++++++++++++++++++++++++++++ CLUSTERING ANALYSIS IS PERFORMING ++++++++++++++++++++++++++
    clmns = ['Gaze point X', 'Gaze point Y']

    df_tr_std = stats.zscore(eye_final[['Gaze point X', 'Gaze point Y']])

    kmeans = KMeans(n_clusters=3, random_state=0).fit(df_tr_std)

    labels = kmeans.labels_
    eye_final['clusters'] = labels
    #Add the column into our list
    clmns.extend(['clusters'])
    #Lets analyze the clusters
    print (eye_final[clmns].groupby(['clusters']).mean())

    sns.lmplot('Gaze point X', 'Gaze point Y',
               data=eye_final,
               fit_reg=False,
               hue="clusters",
               scatter_kws={"marker": "D",
                            "s": 100})
    plt.title('Clusters Wattage vs Duration')
    plt.xlabel('Wattage')
    plt.ylabel('Duration')

    # Defining number of cluster using elbow method
    sse = []
```

```
list_k = list(range(1, 10))

for k in list_k:
    km = KMeans(n_clusters=k)
    km.fit(df_tr_std)
    sse.append(km.inertia_)

# Plot sse against k
plt.figure(figsize=(6, 6))
plt.plot(list_k, sse, '-o')
plt.xlabel(r'Number of clusters *k*')
plt.ylabel('Sum of squared distance');

# defining the silhouette_scores for 2 clusters to 6 clusters
cluster_range = range( 2, 6 )

for n_clusters in cluster_range:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

      # The 1st subplot is the silhouette plot
      # The silhouette coefficient can range from -1, 1 but in this example all
      # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
      # The (n_clusters+1)*10 is for inserting blank space between silhouette
      # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(df_tr_std) + (n_clusters + 1) * 10])

      # Initialize the clusterer with n_clusters value and a random generator
      # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict( df_tr_std )

      # The silhouette_score gives the average value for all the samples.
      # This gives a perspective into the density and separation of the formed
      # clusters
    silhouette_avg = silhouette_score(df_tr_std, cluster_labels)
    print("For n_clusters =", n_clusters, "The average silhouette_score is :", silhouette_avg)

      # Compute the silhouette scores for each sample
```

```
  sample_silhouette_values = silhouette_samples(df_tr_std, cluster_labels)
  y_lower = 10

  for i in range(n_clusters):
    # Aggregate the silhouette scores for samples belonging to
    # cluster i, and sort them
      ith_cluster_silhouette_values = \
      sample_silhouette_values[cluster_labels == i]

      ith_cluster_silhouette_values.sort()

      size_cluster_i = ith_cluster_silhouette_values.shape[0]
      y_upper = y_lower + size_cluster_i

      cmap = plt.cm.viridis
      color = cmap(float(i) / n_clusters)

      ax1.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhouette_values, facecolor=color, edgecolor

        # Label the silhouette plots with their cluster numbers at the middle
      ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
      y_lower = y_upper + 10  # 10 for the 0 samples

  ax1.set_title("The silhouette plot for the various clusters.")
  ax1.set_xlabel("The silhouette coefficient values")
  ax1.set_ylabel("Cluster label")

# The vertical line for average silhoutte score of all the values
  ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

  ax1.set_yticks([])  # Clear the yaxis labels / ticks
  ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

# 2nd Plot showing the actual clusters formed
  colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
  ax2.scatter(df_tr_std[:, 0], df_tr_std[:, 1], marker='.', s=30, lw=0, alpha=0.7,c=colors)

# Labeling the clusters
  centers = clusterer.cluster_centers_
```

```python
        # Draw white circles at cluster centers
        ax2.scatter(centers[:, 0], centers[:, 1], marker='o', c="white", alpha=1, s=200)

        for i, c in enumerate(centers):
            ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1, s=50)

        ax2.set_title("The visualization of the clustered data.")
        ax2.set_xlabel("Feature space for the 1st feature")
        ax2.set_ylabel("Feature space for the 2nd feature")

        plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                      "with n_clusters = %d" % n_clusters),
                     fontsize=14, fontweight='bold')
        plt.show()

else:
    print('THE CLUSTERING ANALYSIS IS SELECTED TO NOT BEING PERFOMRED')
    eye_final['clusters'] = 0

print('++++++++++++++++++++++++++++++++++++++++++++++ FLIGHT DATA ++++++++++++++++++++++++++++++++++++++++++')
print('++++++++++++++++++++++++++++++++++++++++++++++ FLIGHT DATA ++++++++++++++++++++++++++++++++++++++++++')

#####################################################################################################################
#####################################################################################################################
#####################################################################################################################
################################################################## FLIGHT DATA / DATA SYNCHRONIZATION ################
#####################################################################################################################
#####################################################################################################################
#####################################################################################################################

#IMPORTING FILE AND DEFINING THE COLUMN
df_flight = pd.read_csv(filename_flight, delimiter='\t') # RW: added the `delimiter` parameter

if 'alpha,__deg' and '_beta,__deg' not in df_flight:
    df_flight['alpha, deg'] = np.nan
    df_flight['beta, deg'] = np.nan

# THERE IS A CONSTANT NOISE IN THE COLLECTED DATA FROM RUDDER
df_flight['ruddr, yoke1'] = df_flight['ruddr, yoke1'] + 0.0509803891181945

df_flight['timestamp'] = pd.to_datetime(df_flight['timestamp'], format='%m/%d/%Y %H:%M:%S.%f').dt.strftime('%Y-%m-%d %H
```

```
df_flight['timestamp'] = pd.to_datetime(df_flight['timestamp'], format='%Y-%m-%d %H:%M:%S.%f')

rng = pd.date_range(df_flight['timestamp'].dt.round('100ms').iloc[0], periods=(df_flight['timestamp'].iloc[-1] - df_fli

# RW: We will add new rows with the desired time stamps (`date`)
flight_to_append = pd.DataFrame(rng)
flight_to_append.columns = ['timestamp']

# RW: There were some duplicate column names here I had to delete
list_columns = [x for x in df_flight.columns]

dataframe_flight = df_flight[list_columns].copy()
# RW: Mark the existing data as "not stamped"
dataframe_flight['stamped'] = 0

# RW: Add all the necessary empty columns to the new rows
for col in dataframe_flight.columns[1:]:
    flight_to_append[col] = None

# RW: Mark the new (interpolated) data as "stamped"
flight_to_append['stamped'] = 1

# RW: Prepare a new dataframe for interpolation
flight_interpolated = dataframe_flight.append(flight_to_append, ignore_index = True).sort_values('timestamp').set_index
flight_interpolated[['flap, handl']] = flight_interpolated[['flap, handl']].fillna(method='ffill').fillna(method='bfill

#  flight_interpolated_flap = flight_interpolated[['flap, handl', 'stamped']].fillna(method='ffill').fillna(method='bfill
#  flight_interpolated_flap = flight_interpolated_flap.query('stamped == 1').drop('stamped', 1)

# RW: Interpolate! (i.e. fill in data values at the desired moments in time)
# RW: The options used here are a matter of choice, but
#     seemed like a good idea when I examined the data
flight_interpolated.interpolate(limit_direction='both', inplace = True)

#  # RW: This is the data you want
#  df_final = flight_interpolated.query('stamped == 1').drop('stamped', 1)
#
#  df_final = df_final.drop('flap, handl', axis=1)

########################################
## VISUAL CHECK OF WHAT WE'VE DONE!!! ##
```

```
plt.rcParams['figure.figsize'] = 15,6
########################################
plt.rcParams['font.size'] = 18

# RW: For each column, visualize the actual/original
#     values and the interpolated values we've created
#for col in flight_interpolated.columns[1:-1]:
#    flight_interpolated.query('stamped == 0')[col].plot(marker = 'o', lw = 0, markersize = 10)
#    flight_interpolated[col].plot(marker = 'o', lw = 0, markersize = 3)
#    plt.legend(['Actual values', 'Interpolated values'])
#    plt.ylabel(col)
#    plt.show()


synchronized_data = eye_final.merge(flight_interpolated, how='left', left_index=True, right_index=True)

synchronized_data = synchronized_data.reset_index()
synchronized_data['time of scenario'] = synchronized_data['scenario time']
synchronized_data = synchronized_data.set_index('scenario time')



synchronized_data.to_csv('z_10.csv', encoding='utf-8')


# ****************************************************************************************************
# ****************************************************************************************************
# ****************************************************************************************************
# ****************************************************************************************************
# ****************************************************************************************************
# ****************************************************************************************************
# ****************************************************************************************************
# ****************************************************************************************************
#                                   END OF CODING FOR FLIGHT DATA

AOI_list =    ['AI',           'ALT',           'HI',          'TC',          'ASI' ,        'VSI',             'T
synchronized_data['re_entry'] = np.nan
synchronized_data = synchronized_data.reset_index()
```

```
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
############################################################################ RE-ENTRY TIME ################################
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################

print('RE-ENTRY TIME')
for i in range(len(AOI_list)):

    for m in range(0, (synchronized_data.loc[(synchronized_data['fixation'] == 2) & (synchronized_data['AOI'] == AOI_li

        if (m < (synchronized_data.loc[(synchronized_data['fixation'] == 2) & (synchronized_data['AOI'] == AOI_list[i])
            list_index = (synchronized_data.loc[(synchronized_data['fixation'] == 2) & (synchronized_data['AOI'] == AOI

            synchronized_data.at[list_index[m], 're_entry'] = (synchronized_data['scenario time'].iloc[list_index[m+1]]

synchronized_data['rate turn'] = 0.0

##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
############################################################################ RATE OF TURN #################################
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################

print('COMPUTING RATE OF TURN')
i = 0
while i < synchronized_data.index.shape[0]-1:
    synchronized_data.at[i, 'rate turn' ] = (synchronized_data['heading mag'].iloc[i+1] - synchronized_data['heading ma
    i = i + 1

synchronized_data.to_csv('_'  + file_name + '_' + task_name + '_' + 'synchronized_data.csv', encoding='utf-8')
synchronized_data = synchronized_data.set_index('scenario time')

# *** heading objective *** range (25, 35) ==> heading: 30 / range (265, 275) ==> heading: 270
# *** altitude objective *** range (2900, 3100) ==> altitude: 3000 / range (2400, 2600) ==> altitude: 2500

print('COMPUTING DEVIATIONS OF HEADING AND ALTITUDE')
```

```
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
############################################################################ WHAT IS THE FLIGTH SCENARIO ##################
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################

if 25 < synchronized_data['00:00:01.000000':'00:00:02.000000']['heading mag'].mean() < 80:
    objective_heading = 30
else:
    objective_heading = 270

if 2800 < synchronized_data['00:00:01.000000':'00:00:02.000000']['alt, ind'].mean() < 3100:
    objective_altitude = 3000
else:
    objective_altitude = 2500

print('++++++++++++++++++++++++++++++++++++++++++++++++++')
print('objective_heading: ', objective_heading, ', objective_altitude: ', objective_altitude)
print('++++++++++++++++++++++++++++++++++++++++++++++++++')

##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
############################################################################ COMPUTING DEVIATION ##########################
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################

if task_name in {'Landing'}:

    synchronized_data['deviation heading'] = synchronized_data.apply(lambda x: (247.16 - x['heading mag']), axis=1)

else:
    synchronized_data['deviation heading'] = synchronized_data.apply(lambda x: abs(objective_heading - x['heading mag']

synchronized_data['deviation altitude'] = synchronized_data.apply(lambda x: (x['alt, ind'] - objective_altitude) , axis

list_interval = ['00:00:00', '00:00:10', '00:00:20', '00:00:30', '00:00:40', '00:00:50', '00:01:00', '00:01:10', '00:01
```

```
                    '00:03:00', '00:03:10', '00:03:20', '00:03:30', '00:03:40', '00:03:50', '00:04:00']

        synchronized_data['time bracket'] = (synchronized_data['time of scenario']).dt.floor('1S')

        # weeks = [g.reset_index()
        # for n, g in synchronized_data.groupby(pd.TimeGrouper('10s'))]
        # weeks = [g for n, g in synchronized_data.groupby(pd.Grouper(key='timestamp',freq='10s'))]
        # synchronized_data['pitch, deg'].loc[synchronized_data['time bracket'] == '00:01:00'].plot(drawstyle='steps', linewidt
        # synchronized_data['rpm n, engin'].loc[(synchronized_data['time bracket'] >= '00:00:00') & (synchronized_data['time br

        # Estimating True speed using polynomial regression
        # (1) I collected data for pre and post scenaris,(different altitude and different VSI and differnet true speed), then

        print('ESTIMATING TURE AIRSPEED')

        synchronized_data['Vtrue, ktas'] = model.predict(synchronized_data[['Vind, kias', 'alt, ind']])
        synchronized_data['cal rate turn'] = synchronized_data.apply(lambda x: 1091 * math.tan(math.pi * x['roll, deg'] / 180)

        print('ORDERING THE COLUMNS')

        synchronized_data = synchronized_data[['time' ,'timestamp', 'time of scenario', 'time bracket', 'Participant name', 'Sk
                                               'Vind, kias', 'VVI, fpm', 'elev, yoke1', 'ailrn, yoke1', 'ruddr, y
                                               'alpha, deg', 'beta, deg', 'heading mag', 'alt, ind', 'X, m', 'Y,

        print(synchronized_data['Participant name'].unique())


        # find duplicated values
#         print('DUPLICATED VALUES')
#         print(synchronized_data[['Vind, kias', 'VVI, fpm']].loc[synchronized_data[['X, m', 'Y, m', 'Z, m']].duplicated() == Tr
        # ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        #synchronized_data = synchronized_data.drop_duplicates(subset=['X, m', 'Y, m', 'Z, m'], keep='first')
        # ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

        # IT DEFINES IF AN AIRPLANE ENTERS TO THE VICINITY OF AIRPORT.
        # min and max of the airport vicinity
        # dataframe_runway['____X,____m'].min(), dataframe_runway['____X,____m'].max()
        # dataframe_runway['____Z,____m'].min(), dataframe_runway['____Z,____m'].max()
        # dataframe_runway['____Y,____m'].min(), dataframe_runway['____Y,____m'].max()

        print('++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
```

```python
    print(task_name)
    print('+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')

    ##########################################################################################################################
    ##########################################################################################################################
    ##########################################################################################################################
    ################################################################### LOCATION OF LANDING POINT ON RUNWAY ##############
    ##########################################################################################################################
    ##########################################################################################################################
    ##########################################################################################################################
    if task_name == 'Landing':
        print('+++++++++++++++++++++++++++++++++++++++++++++++++++++++ SCENARIO LANDING +++++++++++++++++++++++++++++++++++
        print('SCENARIO IS LANDING')

        origin_runway = [16811.75781, 16449.27734]
        end_runway = [15839.1875, 17182.30078]

        angel_transformation = abs(math.atan((end_runway[1] - origin_runway[1])/(end_runway[0] - origin_runway[0])))

        synchronized_data['X_transformed_airplane'] = synchronized_data.apply(lambda x: origin_runway[0] + math.cos(angel_t

        synchronized_data['Z_transformed_airplane'] = synchronized_data.apply(lambda x: origin_runway[1] + math.sin(angel_t

        # vicinty is based on collected data: it means that airplane enters to the points that data is aviable
        synchronized_data['runway_vicinity'] = synchronized_data.apply(lambda x: 1 if (x['X_transformed_airplane'] > 15500
                                (x['Z_transformed_airplane'] > 16370 and x['Z_transformed_airplane'] < 16500) else 0 , axis

                                #(x['Y, m'] > -31.83329 and x['Y, m'] < -21.41907)

        #dataframe_collected_data['___on,runwy'] = dataframe_runway.apply(lambda x: 1 if x['x_transformed'] > 15593.8841 an
        synchronized_data['___on,runwy'] = synchronized_data.apply(lambda x: 1 if x['X_transformed_airplane'] > 15593.8841
                        x['Z_transformed_airplane'] < 16449.29657 else 0 , axis=1)

        # https://seaborn.pydata.org/generated/seaborn.scatterplot.html
#        ax = sns.scatterplot(x = '____X,____m' , y = '____Z,____m', hue='___on,runwy', size = 0.1 , data = cloud_points_ru
#        ax = sns.scatterplot(x = dataframe_collected_data['X, m'].loc[dataframe_collected_data['runway_vicinity']==1] , y
#                            color = 'black', size = 0.1 , data = dataframe_collected_data)
        #ax.set_xlim([15500, 17500])
        #ax.set_ylim([15500, 17500])
        # https://stackoverflow.com/questions/41517498/how-to-use-scipy-griddata-when-used-with-dataframe-vs-array
```

```
cloud_points_runway['x_transformed'] = cloud_points_runway.apply(lambda x: origin_runway[0] + math.cos(angel_transf


cloud_points_runway['z_transformed'] = cloud_points_runway.apply(lambda x: origin_runway[1] + math.sin(angel_transf
# I used excel to find the boundary of runway based on x and z transfored values
cloud_points_runway['___on,runwy'] = cloud_points_runway.apply(lambda x: 1 if x['x_transformed'] > 15593.8841 and x

# on_runway means that data enters to surrending area of airport (almost 50 meters from the center of z and 200 met
cloud_points_runway = cloud_points_runway.loc[(cloud_points_runway['x_transformed'] > 15500) & (cloud_points_runway

synchronized_data = synchronized_data.reset_index()
# https://stackoverflow.com/questions/38965720/find-closest-point-in-pandas-dataframes

#             if synchronized_data[['X_transformed_airplane', 'Z_transformed_airplane']][synchronized_data[['X_transformed_airpl
#                 synchronized_data = synchronized_data.iloc[:synchronized_data[['X_transformed_airplane', 'Z_transformed_airpla

airplane_point = synchronized_data[['X_transformed_airplane', 'Z_transformed_airplane', 'Y, m']].loc[synchronized_d
airplane_point = airplane_point[['X_transformed_airplane', 'Z_transformed_airplane', 'Y, m']]

could_points = cloud_points_runway[['x_transformed', 'z_transformed', '____Y,____m']]

def closest_point(point, points):
    """ Find closest point from a list of points. """
    return points[cdist([point], points).argmin()]

def match_value(df, col1, x, col2):
    """ Match value x from col1 row to value in col2. """
    return df[df[col1] == x][col2].values[0]

could_points['point'] = [(x, y) for x,y in zip(could_points['x_transformed'], could_points['z_transformed'])]
synchronized_data['point'] = [(x, y) for x,y in zip(synchronized_data['X_transformed_airplane'], synchronized_data[
synchronized_data['closest'] = [closest_point(x, list(could_points['point'])) for x in synchronized_data['point']]
synchronized_data['Y, m cloud'] = [match_value(could_points, 'point', x, '____Y,____m') for x in synchronized_data[

synchronized_data['point'] = synchronized_data.apply(lambda x: 0 if x['runway_vicinity'] != 1 else x['point'], axis
synchronized_data['closest'] = synchronized_data.apply(lambda x: 0 if x['runway_vicinity'] != 1 else x['closest'],
synchronized_data['Y, m cloud'] = synchronized_data.apply(lambda x: 0 if x['runway_vicinity'] != 1 else x['Y, m clo

synchronized_data['runway_to_airplane'] = synchronized_data.apply(lambda x: x['Y, m'] - x['Y, m cloud'], axis=1)

threshold_landing = 0.15
```

```
synchronized_data['landed or not'] = synchronized_data.apply(lambda x: 1 if x['runway_to_airplane'] <= threshold_la

#synchronized_data['x optimal landing point'] = 16671
synchronized_data['y optimal landing point'] = -21.916
#synchronized_data['z optimal landing point'] = 16536

synchronized_data['x optimal landing point'] = cloud_points_runway.apply(lambda x: origin_runway[0] + math.cos(ange
synchronized_data['z optimal landing point'] = cloud_points_runway.apply(lambda x: origin_runway[1] + math.sin(ange

synchronized_data['x landing point'] = synchronized_data['X_transformed_airplane'].loc[synchronized_data['landed or
synchronized_data['z landing point'] = synchronized_data['Z_transformed_airplane'].loc[synchronized_data['landed or

synchronized_data['x deviation landing'] = synchronized_data['x optimal landing point'].iloc[0] - synchronized_data

synchronized_data['z deviation landing'] = synchronized_data['z optimal landing point'].iloc[0] - synchronized_data

synchronized_data['status of landing'] = synchronized_data.apply(lambda x: 'landed after optimal point' if x['x dev

style.available
#style.use('ggplot')
style.use('classic') #sets the size of the charts
#style.use('ggplot')
# first bmh and then classic
# styles classic , bmh , seaborn-bright, seaborn-colorblind, seaborn-deep
# https://python-graph-gallery.com/
# https://tonysyu.github.io/raw_content/matplotlib-style-gallery/gallery.html
# https://stackoverflow.com/questions/11640243/pandas-plot-multiple-y-axes
formatter = mticker.ScalarFormatter(useMathText=True)
formatter.set_powerlimits((-3,2))

plt.rcParams.update({'font.size': 9})
plt.rcParams["font.weight"] = "bold"
plt.rcParams["axes.labelweight"] = "bold"

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(17, 7.5), sharex=True)

#axes.scatter(x='x_transformed' , y= 'z_transformed', c='___on,runwy', data = cloud_points_runway)
L1 = axes.vlines(15593.8841, ymin=16418.98266 , ymax=16449.29657, linestyle='-', linewidth=2.0, color='mediumseagre
L2 = axes.vlines(16814.52156, ymin=16418.98266 , ymax=16449.29657, linestyle='-', linewidth=2.0, color = 'mediumsea
L3 = axes.hlines(16418.98266, xmin=15593.8841 , xmax=16814.52156 , linestyle='-', linewidth=2.0, color = 'mediumsea
L4 = axes.hlines(16449.29657, xmin=15593.8841 , xmax=16814.52156 , linestyle='-', linewidth=2.0, color = 'mediumsea
```

```
            L6 = sns.scatterplot(ax = axes, x='x_transformed' , y= 'z_transformed', hue='___on,runwy', size = 0.1 , data = clou
            L7 = axes.scatter(x=synchronized_data['X_transformed_airplane'].iloc[synchronized_data['X_transformed_airplane'].lo
                              y=synchronized_data['Z_transformed_airplane'].iloc[synchronized_data['X_transformed_airplane'].lo
            L8 = axes.scatter(synchronized_data['x optimal landing point'].iloc[0], synchronized_data['z optimal landing point'
            axes.scatter(synchronized_data['x landing point'].max(), synchronized_data['z landing point'].max(), s=150, facecol

            axes.legend(loc="upper right")
            axes.legend([L1, L7, L8],['Runway', 'Flight path', 'Touchdown Zone'], framealpha = 0.7)
            axes.set_xlabel('X (meter)')
            axes.set_ylabel('Y (meter)')

            axes.grid()
            plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)

            plt.savefig('_'  + file_name + '_' + task_name + '.png', bbox_inches='tight')
            plt.savefig('_'  + file_name + '_' + task_name + '.pdf', bbox_inches='tight')
            plt.show()

            synchronized_data = synchronized_data.set_index('scenario time')

        else:
            list_landing_columns  = ['X_transformed_airplane', 'Z_transformed_airplane', 'runway_vicinity', '___on,runwy', 'poi

            for item in list_landing_columns:
                synchronized_data[item] = 0
        # PLOTING SCYCHRONIZE DATA OF EYE MOVEMENT DATA WITH FLIGHT DATA
        # synchronized_data['AI'].loc[(synchronized_data['time bracket'] >= list_interval[0]) & (synchronized_data['time bracke
        # VERSION 294 ==> ADDING INPUT TO THE CHART
        flight_list = ['pitch, deg', 'roll, deg', 'deviation altitude' , 'deviation heading' , 'cal rate turn' , 'Vind, kias' ,


        if type(synchronized_data.index) == pd.core.indexes.range.RangeIndex:
            time.sleep(0.0)
#            print('STEP 1: INDEX IS A RANGE AND IT SHOULD BE')
        else:
            synchronized_data = synchronized_data.reset_index()

        synchronized_data['trim, elev_pct_change'] = 0
        synchronized_data['singal_trim, elev_pct_change'] = 0
        synchronized_data['trim, elev_pct_change'] = synchronized_data['trim, elev'].pct_change()
```

```
synchronized_data['trim, elev_pct_change'] = synchronized_data['trim, elev_pct_change'].fillna(0)

signal_counter = 1
signal_flag_counter = False
for index, row in synchronized_data.iterrows():

    if index == 0:
        continue
    elif index == synchronized_data.index[-1]:
        break
    else:

        if synchronized_data['trim, elev_pct_change'].iloc[index] == 0 and synchronized_data['trim, elev_pct_change'].i
            synchronized_data['singal_trim, elev_pct_change'].iloc[index+1] = signal_counter
            signal_flag_counter = True

        if synchronized_data['trim, elev_pct_change'].iloc[index] != 0 and signal_flag_counter == True:
            synchronized_data['singal_trim, elev_pct_change'].iloc[index] = signal_counter

        if synchronized_data['trim, elev_pct_change'].iloc[index-1] != 0 and synchronized_data['trim, elev_pct_change']
            signal_flag_counter = False
            signal_counter += 1



##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
############################################################################ SETTING VARIABLES OF SIGNAL DETECTION #######
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
threshold_signal = 0.060
threshold_signal_std = 0.01
threshold_prominence = 0.6
threshold_height = 0.01
threshold_distace = 30
threshold_duration = 0.5
threshold_point = 10
threshold_signal_max = 0.075
threshold_width = 0.01
```

```
# rolling based on 6 points
window = 5
# Rolling type
agg = 'std'
#######################################################################################################################
#######################################################################################################################
#######################################################################################################################
######################################################################### SETTING VARIABLES OF SIGNAL DETECTION #######
#######################################################################################################################
#######################################################################################################################
#######################################################################################################################

#######################################################################################################################
#######################################################################################################################
#######################################################################################################################
################################################################### DETECTION OF SIGNALS' PEAKS AND LOOKING AT THE PEAK
#######################################################################################################################
#######################################################################################################################
#######################################################################################################################
# ALL POINTS OF elev TAKEN FRON PANDAS DATAFRAME / ALL POINTS OF ailrn TAKEN FRON PANDAS DATAFRAME
x_elev = synchronized_data['elev, yoke1'].values
x_ailrn = synchronized_data['ailrn, yoke1'].values
x_ruddr = synchronized_data['ruddr, yoke1'].values
# MAX AND MIN OF SIGNALS (elev) / peaks_elev INDEX OF MAX POINTS ==> elev /  _peaks_elev INDEX OF MIN POINTS ==> elev

list_peaks_points = [x_elev, x_ailrn, x_ruddr]

#######################################################################################################################
#######################################################################################################################
#######################################################################################################################
################################################################### DETECTION OF SIGNALS' PEAKS AND PLOTTING DATA #####
#######################################################################################################################
#######################################################################################################################
#######################################################################################################################

#######################################################################################################################
#######################################################################################################################
#######################################################################################################################
################################################################### ADDING THRESHOLD AND REMOVING NOISES FROM SIGNAL ##
#######################################################################################################################
#######################################################################################################################
```

```
#########################################################################################################################

sig_detect_list = ['signal_ailrn_detect', 'signal_elev_detect', 'signal_ruddr_detect']
sig_list = ['ailrn, yoke1', 'elev, yoke1', 'ruddr, yoke1']

for index, member in enumerate(sig_detect_list):
    if member in {'signal_elev_detect'}:

        synchronized_data[member] = synchronized_data.apply(lambda x: 0 if (-threshold_signal < x[sig_list[index]] < th

        synchronized_data[member] = synchronized_data.apply(lambda x: 1 if (x['trim, elev_pct_change'] != 0) else x[mem
    else:
        synchronized_data[member] = synchronized_data.apply(lambda x: 0 if (-threshold_signal < x[sig_list[index]] < th

# I ADDED THIS LINE, BEACAUSE IT DOES NOT DETECT PART OF A SIGNAL THAT STARTS AS 1 NO 0
    if synchronized_data[member].iloc[0] == 1: synchronized_data[member].iloc[0] = 0
    if synchronized_data[member].iloc[-1] == 1: synchronized_data[member].iloc[-1] = 0

number_of_row = 10
col_name = ['ailrn, yoke1', 'elev, yoke1', 'ruddr, yoke1']
col_detection_name = ['signal_ailrn_detect', 'signal_elev_detect', 'signal_ruddr_detect' ]

row_index = 0
for row in synchronized_data.iterrows():
    for col_index in range(0,3):

        if row_index > number_of_row and row_index < (synchronized_data.shape[0]-1)-number_of_row:
            if         (synchronized_data[col_name[col_index]].iloc[row_index - 9 : row_index+1].loc[synchronized_data[
            synchronized_data[col_name[col_index]].iloc[row_index+1:row_index+threshold_point].loc[synchronized_data[co
            synchronized_data[col_name[col_index]].iloc[row_index] > threshold_signal and synchronized_data[col_name[co
                        (synchronized_data[col_name[col_index]].iloc[row_index - 9 : row_index+1].loc[synchronized_data
            synchronized_data[col_name[col_index]].iloc[row_index+1:row_index+threshold_point].loc[synchronized_data[co
            synchronized_data[col_name[col_index]].iloc[row_index] < -threshold_signal and synchronized_data[col_name[c

                synchronized_data[col_name[col_index]].iloc[row_index] = 0
                synchronized_data[col_detection_name[col_index]].iloc[row_index] = 0

    row_index += 1

x_elev_std = synchronized_data['elev, yoke1'].rolling(window).agg([agg]).values
x_elev_std = np.asarray(x_elev_std).squeeze()
```

```
x_elev_std = np.nan_to_num(x_elev_std)

x_ailrn_std = synchronized_data['ailrn, yoke1'].rolling(window).agg([agg]).values
x_ailrn_std = np.asarray(x_ailrn_std).squeeze()
x_ailrn_std = np.nan_to_num(x_ailrn_std)

x_ruddr_std = synchronized_data['ruddr, yoke1'].rolling(window).agg([agg]).values
x_ruddr_std = np.asarray(x_ruddr_std).squeeze()
x_ruddr_std = np.nan_to_num(x_ruddr_std)

# MAX OF elev STANDARD DEVIATION ==> THERE IS NO "MIN" FOR STANDARD DEVIATION OF elev and ailrn
peaks_elev_std, _elev_std = find_peaks(x_elev_std,  height = threshold_height, distance = threshold_distace)
_peaks_elev_std, _p_elev_std = find_peaks(-x_elev_std, height = threshold_height, distance = threshold_distace)

# MAX OF ailrn STANDARD DEVIATION  == > WAS ONE
peaks_ailrn_std, _ailrn_std = find_peaks(x_ailrn_std, prominence = (None, threshold_prominence), height = threshold_hei
_peaks_ailrn_std, _p_ailrn_std = find_peaks(-x_ailrn_std, prominence = (None, threshold_prominence), height = threshold

peaks_ruddr_std, _ruddr_std = find_peaks(x_ruddr_std, prominence = (None, threshold_prominence), height = threshold_hei
_peaks_ruddr_std, _p_ruddr_std = find_peaks(-x_ruddr_std, prominence = (None, threshold_prominence), height = threshold

list_peaks = [peaks_ailrn_std, peaks_elev_std, peaks_ruddr_std]
list_peaks_col = ['signal_ailrn_max_std', 'signal_elev_max_std', 'signal_ruddr_max_std']
list_peaks_points = [x_ailrn_std, x_elev_std, x_ruddr_std]

if len(peaks_elev_std) > 1:
    for i in range(peaks_elev_std.shape[0]):
        synchronized_data.at[peaks_elev_std[i], 'signal_elev_max_std'] =  x_elev_std[peaks_elev_std[i]]
else:
    synchronized_data['signal_elev_max_std'] = 0

if len(peaks_ailrn_std) > 1:
    for i in range(peaks_ailrn_std.shape[0]):
        synchronized_data.at[peaks_ailrn_std[i], 'signal_ailrn_max_std'] =  x_ailrn_std[peaks_ailrn_std[i]]
else:
    synchronized_data['signal_ailrn_max_std'] = 0

if len(peaks_ruddr_std) > 1:
    for i in range(peaks_ruddr_std.shape[0]):
        synchronized_data.at[peaks_ruddr_std[i], 'signal_ruddr_max_std'] =  x_ruddr_std[peaks_ruddr_std[i]]
else:
```

```
    synchronized_data['signal_ruddr_max_std'] = 0

style.available
#style.use('ggplot')
style.use('classic') #sets the size of the charts

plt.rcParams.update({'font.size': 9})
plt.rcParams["font.weight"] = "bold"
plt.rcParams["axes.labelweight"] = "bold"

fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(19, 10), sharex=True)

for i in range(3):
    axes[i].xaxis.grid(True)
    axes[i].minorticks_on()
    axes[i].xaxis.grid(True, which="both")
    axes[i].grid(b=True, axis='both', which='minor', linestyle=':', linewidth='0.5', color='black')
    axes[i].grid(b=True, axis='both', which='major', linestyle='--', linewidth='0.5', color='black')
    axes[i].set_xlim([0, synchronized_data.shape[0]])
    axes[i].set_ylim([-1.015, 1.015])
    # IF DOES NOT MENTION THIS LINES OF CODE, THEN IT WILL NOT PLOT ANYTHINGS

synchronized_data['elev, yoke1'].plot(ax=axes[0], linestyle= '-' , linewidth = 1, style='b-', marker='+', c='b', ms = 4
synchronized_data['signal_elev_detect'].plot(ax=axes[0], linestyle= '-', drawstyle='steps', linewidth = 2, c='g')
synchronized_data['elev, yoke1'].rolling(5).agg(['std']).plot(ax=axes[0], linestyle= '-', linewidth = 1, style='b-', ma

if 'signal_elev_max_std' in synchronized_data.columns:
    if synchronized_data['signal_elev_max_std'].count() < 50:
        synchronized_data['signal_elev_max_std'].plot(ax=axes[0], style='.', marker = '*', c='black', ms = 10)
    else:
        synchronized_data['signal_elev_max_std'].plot(ax=axes[2], style='.', marker = '*', c='black', ms = 2)

synchronized_data['ailrn, yoke1'].plot(ax=axes[1], linestyle= '-' , linewidth = 1, style='b-', marker='+', c='b', ms =
synchronized_data['signal_ailrn_detect'].plot(ax=axes[1], linestyle= '-', drawstyle='steps', linewidth = 2, c='g')
synchronized_data['ailrn, yoke1'].rolling(5).agg(['std']).plot(ax=axes[1], linestyle= '-', linewidth = 1, style='b-', m

if 'signal_ailrn_max_std' in synchronized_data.columns:
    if synchronized_data['signal_ailrn_max_std'].count() < 50:
        synchronized_data['signal_ailrn_max_std'].plot(ax=axes[1], style='.', marker = '*', c='black', ms = 10)
    else:
        synchronized_data['signal_ailrn_max_std'].plot(ax=axes[2], style='.', marker = '*', c='black', ms = 2)
```

```
synchronized_data['ruddr, yoke1'].plot(ax=axes[2], linestyle= '-' , linewidth = 1, style='b-', marker='+', c='b', ms = 
synchronized_data['signal_ruddr_detect'].plot(ax=axes[2], linestyle= '-', drawstyle='steps', linewidth = 2, c='g')
synchronized_data['ruddr, yoke1'].rolling(5).agg(['std']).plot(ax=axes[2], linestyle= '-', linewidth = 1, style='b-', m

if 'signal_ruddr_max_std' in synchronized_data.columns:
    if synchronized_data['signal_ruddr_max_std'].count() < 50:
        synchronized_data['signal_ruddr_max_std'].plot(ax=axes[2], style='.', marker = '*', c='black', ms = 10)
    else:
        synchronized_data['signal_ruddr_max_std'].plot(ax=axes[2], style='.', marker = '*', c='black', ms = 2)

for i in range(3):
    axes[i].legend(loc='lower right', framealpha = 0.6)
    axes[i].grid()

plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)

plt.savefig('_'  + file_name + '_' + task_name + '_'  + 'primary_signal.png', bbox_inches='tight')
plt.savefig('_'  + file_name + '_' + task_name + '_'  + 'primary_signal.pdf', bbox_inches='tight')

plt.show()

############################################################                F I R S T CHART                    #######
############################################################                F I R S T CHART                    #######
############################################################                F I R S T CHART                    #######
############################################################                F I R S T CHART                    #######
############################################################                F I R S T CHART                    #######

#######################################################################################################################
#######################################################################################################################
#######################################################################################################################
#################################################################### ADDING THRESHOLD AND REMOVING NOISES FROM SIGNAL ##
#######################################################################################################################
#######################################################################################################################
#######################################################################################################################
synchronized_data.to_csv('_'  + file_name + '_' + task_name + '_STEP_1.csv', encoding='utf-8')
#######################################################################################################################
#######################################################################################################################
#######################################################################################################################
################################################################### DETECT SIGNAL, MERGE THEM AND REMOVE NOISE ONES ###
###################################################################                  S T E P NO.2                  ###
```

```
##########################################################################################################################
##########################################################################################################################

col_name = ['ailrn, yoke1', 'elev, yoke1', 'ruddr, yoke1']
col_detection_name = ['signal_ailrn_detect', 'signal_elev_detect', 'signal_ruddr_detect' ]
sig_pre_list = ['signal_ailrn_previous_flag', 'signal_elev_previous_flag', 'signal_ruddr_previous_flag']
sig_cur_list = ['signal_ailrn_current_flag','signal_elev_current_flag','signal_ruddr_current_flag']
sig_counter_list =['signal_counter_ailrn', 'signal_counter_elev', 'signal_counter_ruddr']
sig_start_time_list =['signal_start_time_ailrn', 'signal_start_time_elev', 'signal_start_time_ruddr']
sig_duration_list = ['signal_duration_ailrn', 'signal_duration_elev', 'signal_duration_ruddr']
sig_between_list = ['signal_timebetween_ailrn', 'signal_timebetween_elev', 'signal_timebetween_ruddr']

list_parameters = sig_pre_list + sig_cur_list + sig_counter_list + sig_start_time_list + sig_duration_list + sig_betwee

for member in list_parameters:
    synchronized_data[member] = np.nan

for index, member in enumerate(col_name):
    # INDEX OF START OF SIGNAL
    signal_index = 0
    # SIGNAL COUNTER
    signal_counter = 0
    # ROW COUNTER
    row_index = 0
    # signal duration
    signal_duration = 0
    signal_timebetween = 0
    merge_flag = False

    for row in synchronized_data.iterrows():
        # IF ROW INDEX IS 0 THEN ADDING ONE TO ROW INDEX AND SKIPPING TO NEXT LOOP
        if row_index == 0:
            row_index += 1
            continue
        # AT THE END OF PANDAS DATAFRAME; (1) IT PUTS THE END TIME, & (2) COMPUTES THE SIGNAL DURATION TIME, (3) COMPUT
        elif row_index == synchronized_data.index[-1]:
            if synchronized_data[col_detection_name[index]].iloc[row_index-1] == 1:

                synchronized_data.at[row_index, sig_start_time_list[index]] = synchronized_data['time of scenario'].ilo
                end_time = synchronized_data['time of scenario'].iloc[row_index]
                synchronized_data.at[signal_index, sig_duration_list[index]] = (end_time - start_time) /np.timedelta64(
```

```
            signal_duration = (end_time - start_time) /np.timedelta64(1, 's')
        break

    # IF THIS IS NOT SIGNAL THEN NOTHING
    #             elif synchronized_data['signal_ailrn_detect'].iloc[row_index] == 0 and synchronized_data['signal_a
    #                 signal_event = False
    # IF THIS IS SIGNAL, THEN IT RECORDS (1) THE START TIME OF SIGNAL, (2) ADD ONE TO SIGNAL COUNTER VARIABLE (3) C
    ##########################################################################################################
    ########################################################################## ENTER A SIGNAL ###############
    ##########################################################################################################
    elif synchronized_data[col_detection_name[index]].iloc[row_index-1] == 0 and synchronized_data[col_detection_na
        time.sleep(0.0)
        synchronized_data.at[row_index, sig_start_time_list[index]] = synchronized_data['time of scenario'].iloc[ro
        synchronized_data.at[row_index, sig_counter_list[index]] = signal_counter + 1
        start_time = synchronized_data['time of scenario'].iloc[row_index]
        signal_index = row_index
        signal_counter += 1

        if signal_counter  == 1 and synchronized_data[col_name[index]].iloc[row_index] > 0:
            signal_previous_flag = True
            synchronized_data[sig_pre_list[index]].iloc[row_index] = signal_previous_flag

        elif signal_counter == 1 and synchronized_data[col_name[index]].iloc[row_index] < 0:

            signal_previous_flag = False
            synchronized_data[sig_pre_list[index]].iloc[row_index] = signal_previous_flag

        elif signal_counter == 2 and synchronized_data[col_name[index]].iloc[row_index] > 0:

            signal_current_flag = True
            synchronized_data[sig_pre_list[index]].iloc[row_index] = signal_previous_flag
            synchronized_data[sig_cur_list[index]].iloc[row_index] = signal_current_flag

        elif signal_counter == 2 and synchronized_data[col_name[index]].iloc[row_index] < 0:
            signal_current_flag = False

            synchronized_data[sig_pre_list[index]].iloc[row_index] = signal_previous_flag
            synchronized_data[sig_cur_list[index]].iloc[row_index] = signal_current_flag

        elif signal_counter > 2 and synchronized_data[col_name[index]].iloc[row_index] > 0:
            signal_previous_flag = signal_current_flag
```

```
            signal_current_flag = True

            synchronized_data[sig_pre_list[index]].iloc[row_index] = signal_previous_flag
            synchronized_data[sig_cur_list[index]].iloc[row_index] = signal_current_flag

        elif signal_counter > 2 and synchronized_data[col_name[index]].iloc[row_index] < 0:
            signal_previous_flag = signal_current_flag
            signal_current_flag = False

            synchronized_data[sig_pre_list[index]].iloc[row_index] = signal_previous_flag
            synchronized_data[sig_cur_list[index]].iloc[row_index] = signal_current_flag

        # IT COMPUTES THE TIME BETWEEN OF SIGNALS IF THIS IS SECOND SIGNAL; THEN IT COMPUTES TIME BETWEEN WHICH IS
        ##########################################################################################################
        ############################################################## TIME BETWEEN OF SIGNALS ###################
        ##########################################################################################################
        ##########################################################################################################
        ############################################################## MERGING SIGNALS ###########################
        ##########################################################################################################
        if signal_counter > 1 and row_index > 10:
            synchronized_data.at[row_index, sig_between_list[index]] = (start_time - end_time) / np.timedelta64(1,
            signal_timebetween = (start_time - end_time) / np.timedelta64(1, 's')

            if synchronized_data[col_detection_name[index]].iloc[row_index - threshold_point : row_index].loc[synch
            ((signal_current_flag == True and signal_previous_flag == True) or \
             (signal_current_flag == False and signal_previous_flag == False)):
                counter_reverse = signal_index - 1
                signal_counter -= 1

                synchronized_data.at[signal_index, sig_counter_list[index]] = np.nan
                synchronized_data.at[signal_index, sig_start_time_list[index]] = np.nan
                # MOVING BACKWARD ==> WHERE SIGNALS SHOULD BE MERGED
                while synchronized_data[col_detection_name[index]].iloc[counter_reverse] == 0:
                    synchronized_data.at[counter_reverse, col_detection_name[index]] = 1
                    synchronized_data.at[counter_reverse, sig_start_time_list[index]] = np.nan

                    counter_reverse -= 1
                merge_flag = True
    ##########################################################################################################
    ############################################################## THE END OF A SIGNAL #######################
    ##########################################################################################################
```

```
elif synchronized_data[col_detection_name[index]].iloc[row_index-1] == 1 and synchronized_data[col_detection_na
    # it gets the time the signal is over
    synchronized_data.at[row_index, sig_start_time_list[index]] = synchronized_data['time of scenario'].iloc[ro
    # the end_time is computed
    end_time = synchronized_data['time of scenario'].iloc[row_index]
    # signal dusration is computed usign end time and start time
    # I use signal index to put the signal duration in front of the first row of signal
    synchronized_data.at[signal_index, sig_duration_list[index]] = (end_time - start_time) /np.timedelta64(1, '
    signal_duration = (end_time - start_time) /np.timedelta64(1, 's')
##################################################################################################################
##################################################################### REMOVING FIRST SIGNAL IF DURATION IS LESS T
##################################################################################################################
# IF THIS IS THE FIRST SIGNAL AND SIGNAL DURATION IS < 0.5 OR IF THIS IS NOT FIRST SIGNAL, BUT TIME BETWEEN SIG
    if (signal_counter == 1 and
        row_index > 10 and
        signal_duration < threshold_duration and
        abs(synchronized_data[col_name[index]].iloc[signal_index:row_index]).max() < threshold_signal_max and
        merge_flag == False and
        synchronized_data[col_detection_name[index]].iloc[row_index:row_index + threshold_point].loc[synchroniz
        (signal_counter > 1 and
         (synchronized_data[col_detection_name[index]].iloc[signal_index - threshold_point : signal_index].loc[
          synchronized_data[col_detection_name[index]].iloc[row_index:row_index + threshold_point].loc[synchron
          signal_duration < threshold_duration and
          abs(synchronized_data[col_name[index]].iloc[signal_index:row_index]).max() < threshold_signal_max):
        signal_counter -= 1

        synchronized_data.at[signal_index, sig_counter_list[index]] = np.nan
        synchronized_data.at[signal_index, sig_start_time_list[index]] = np.nan

        counter_reverse = row_index - 1

        while synchronized_data[col_detection_name[index]].iloc[counter_reverse] == 1:

            time.sleep(0.0)
            synchronized_data[col_detection_name[index]].iloc[counter_reverse] = 0
            synchronized_data[sig_counter_list[index]].iloc[signal_index] = np.nan

            counter_reverse -= 1
    merge_flag = False
```

```python
        row_index += 1

    synchronized_data[sig_counter_list[index]] = synchronized_data.groupby(col_detection_name[index])[sig_counter_list[

synchronized_data.to_csv('_'  + file_name + '_' + task_name + '_STEP_BEFORE.csv', encoding='utf-8')
#######################################################################################################################
#######################################################################################################################
#######################################################################################################################
################################################################### TIME THRESH FOR SIGNAL DETECTION #################
#######################################################################################################################
#######################################################################################################################
#######################################################################################################################
#----------------------------------------------------------------------------------------------------------------------

style.available
style.use('classic')

plt.rcParams.update({'font.size': 9})
plt.rcParams["font.weight"] = "bold"
plt.rcParams["axes.labelweight"] = "bold"

fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(19, 10), sharex=True)
for i in range(3):

    axes[i].xaxis.grid(True)
    axes[i].minorticks_on()
    axes[i].xaxis.grid(True, which="both")
    axes[i].grid(b=True, axis='both', which='minor', linestyle=':', linewidth='0.5', color='black')
    axes[i].grid(b=True, axis='both', which='major', linestyle='--', linewidth='0.5', color='black')

    axes[i].set_ylim([-1.015, 1.015])
    axes[i].set_xlim([0, synchronized_data.shape[0]])
    axes[i].grid()
    axes[i].axhline(y= -0.01, linestyle = '--', color='g')
    axes[i].axhline(y= 0.01, linestyle = '--', color='g')

synchronized_data['elev, yoke1'].plot(ax=axes[0], linestyle= '-' , linewidth = 1, style='b-', marker='+', c='b', ms = 4
synchronized_data['elev, yoke1'].rolling(window).agg([agg]).plot(ax=axes[0], linestyle= '-' , linewidth = 2, style='b-'
synchronized_data['signal_elev_detect'].plot(ax=axes[0], linestyle= '-', drawstyle='steps', linewidth = 2, c='g', zorde
if synchronized_data['signal_elev_max_std'].count() < 50:
    synchronized_data['signal_elev_max_std'].plot(ax=axes[0], style='.', marker = '*', c='black', ms = 8, zorder=3)
```

```
else:
    synchronized_data['signal_elev_max_std'].plot(ax=axes[0], style='.', marker = '*', c='black', ms = 2, zorder=3)

synchronized_data['ailrn, yoke1'].plot(ax=axes[1], linestyle= '-' , linewidth = 1, style='b-', marker='+', c='b', ms = 
synchronized_data['ailrn, yoke1'].rolling(window).agg([agg]).plot(ax=axes[1], linestyle= '-' , linewidth = 2, style='b-
synchronized_data['signal_ailrn_detect'].plot(ax=axes[1], linestyle= '-', drawstyle='steps', linewidth = 2, c='g', zord

if synchronized_data['signal_ailrn_max_std'].count() < 50:
    synchronized_data['signal_ailrn_max_std'].plot(ax=axes[1], style='.', marker = '*', c='black', ms= 8, zorder=3)
else:
    synchronized_data['signal_ailrn_max_std'].plot(ax=axes[1], style='.', marker = '*', c='black', ms= 2, zorder=3)

synchronized_data['ruddr, yoke1'].plot(ax=axes[2], linestyle= '-' , linewidth = 1, style='b-', marker='+', c='b', ms = 
synchronized_data['ruddr, yoke1'].rolling(window).agg([agg]).plot(ax=axes[2], linestyle= '-' , linewidth = 2, style='b-
synchronized_data['signal_ruddr_detect'].plot(ax=axes[2], linestyle= '-', drawstyle='steps', linewidth = 2, c='g', zord

if synchronized_data['signal_ruddr_max_std'].count() < 50:
    synchronized_data['signal_ruddr_max_std'].plot(ax=axes[2], style='.', marker = '*', c='black', ms= 8, zorder=3)
else:
    synchronized_data['signal_ruddr_max_std'].plot(ax=axes[2], style='.', marker = '*', c='black', ms= 2, zorder=3)


for i in range(3):
    axes[i].legend(loc='lower right', framealpha = 0.6)
    axes[i].grid()

plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)

plt.savefig('_'  + file_name + '_' + task_name + '_'  + 'secondary_signal.png', bbox_inches='tight')
plt.savefig('_'  + file_name + '_' + task_name + '_'  + 'secondary_signal.pdf', bbox_inches='tight')

plt.show()

############################################################              S E C O N D CHART                 #######
############################################################              S E C O N D CHART                 #######
############################################################              S E C O N D CHART                 #######
############################################################              S E C O N D CHART                 #######
############################################################              S E C O N D CHART                 #######


synchronized_data.to_csv('_'  + file_name + '_' + task_name + '_STEP_2.csv', encoding='utf-8')
```

```python
#################################################################################################################################
#################################################################################################################################
#################################################################################################################################
################################################################### MOVING STANDARD DEVIATIONS POINTS TO INSIDE #######
###################################################################                STEP 3                           #######
#################################################################################################################################
#################################################################################################################################

col_name = ['ailrn, yoke1', 'elev, yoke1', 'ruddr, yoke1']
col_detection_name = ['signal_ailrn_detect', 'signal_elev_detect', 'signal_ruddr_detect' ]
list_peaks_col = ['signal_ailrn_max_std', 'signal_elev_max_std', 'signal_ruddr_max_std']
sig_pre_list = ['signal_ailrn_previous_flag', 'signal_elev_previous_flag', 'signal_ruddr_previous_flag']
sig_cur_list = ['signal_ailrn_current_flag','signal_elev_current_flag','signal_ruddr_current_flag']
sig_counter_list =['signal_counter_ailrn', 'signal_counter_elev', 'signal_counter_ruddr']
sig_start_time_list =['signal_start_time_ailrn', 'signal_start_time_elev', 'signal_start_time_ruddr']
sig_duration_list = ['signal_duration_ailrn', 'signal_duration_elev', 'signal_duration_ruddr']
sig_between_list = ['signal_timebetween_ailrn', 'signal_timebetween_elev', 'signal_timebetween_ruddr']
sig_max_std = ['t_signal_ailrn_max_std', 't_signal_elev_max_std', 't_signal_ruddr_max_std']

for index, member in enumerate(col_name):
    synchronized_data[sig_max_std[index]] = np.nan
    synchronized_data[sig_max_std[index]] = synchronized_data.apply(lambda x: 1 if x[list_peaks_col[index]] > 0 else np

    signal_potensial = synchronized_data[sig_counter_list[index]].dropna().unique().tolist()

    for i in signal_potensial:
        list_signal = (synchronized_data[sig_counter_list[index]].loc[synchronized_data[sig_counter_list[index]] == i])
        # CHECKING NUMBER OF POINTS IN A SIGNAL TO BE MORE THAN 2
        # CHECKING BACKWARD
        # IT IS THE INDEX OF THE FIRST STANDARD DEVIATION
        index_s = synchronized_data[sig_counter_list[index]].loc[synchronized_data[sig_counter_list[index]] == i].index
        # IT COUNTS THE NUMBER OF MAX STANDARD DEVIATION THAT SHOULD BE STUDIED FOR COUNTED SIGNALS
        if synchronized_data[list_peaks_col[index]].loc[synchronized_data[sig_counter_list[index]].loc[synchronized_dat
            # THERE A STANDARD DEVIATION IN LAST 4 ROWS AND EMPTY CELLS TO IN THE FIRST 3 ROWS OF THE RANGE ==> IF YES,
            time.sleep(0)
            if pd.notna(synchronized_data[list_peaks_col[index]].iloc[index_s : index_s + 3]).any() == False and \
            pd.notna(synchronized_data[list_peaks_col[index]].iloc[index_s - 4 : index_s]).any() == True:
                # IF 1ST BEFORE IS NOT NAN VALUE
                synchronized_data.at[index_s, sig_max_std[index]] = 1
                # IF 1ST BEFORE IS NOT NAN VALUE
            else:
```

```
            time.sleep(0.0)
        # NOW CHECKING THE LAST ROW ==> IF IT GETS TI LAST ROWS IT WILL STOP IT
        if synchronized_data.index[-1] <= synchronized_data[sig_counter_list[index]].loc[synchronized_data[sig_coun
            break
        else:
            index_e = synchronized_data[sig_counter_list[index]].loc[synchronized_data[sig_counter_list[index]] ==
            # OTHERWISE IT CHECKES LAST 2 RECORDS ARE EMPTY AND IF THERE IS A VALUE IN THE NEXT 4 ROWS ==> IF YES,
            if pd.notna(synchronized_data[list_peaks_col[index]].iloc[index_e - 2 : index_e + 1]).any() == False an
            pd.notna(synchronized_data[list_peaks_col[index]].iloc[index_e + 1 : index_e + 5]).any() == True:
                # IF 1ST BEFORE IS NOT NAN VALUE
                synchronized_data.at[index_e, sig_max_std[index]] = 1
            else:
                time.sleep(0.0)
    else:
        time.sleep(0.0)
    # IF THERE ARE THREE PEAKS OF STANDARD DEVIATION THEN THE RANGE SHOULD BE CHECKED / 4 POINTS BEFORE AND AFTER
    # HAVING AT LEAST 2 SECTIONS
sum_factor = 0

for i in signal_potensial:
    if synchronized_data[sig_max_std[index]].loc[synchronized_data[sig_counter_list[index]].loc[synchronized_data[s
        sum_factor += synchronized_data[sig_max_std[index]].loc[synchronized_data[sig_counter_list[index]].loc[sync

for i in range(1, sum_factor+1):
    signal_potensial.append(signal_potensial[-1]+1)

for i in signal_potensial:
    if synchronized_data[sig_max_std[index]].loc[synchronized_data[sig_counter_list[index]].loc[synchronized_data[s
        # GETTING THE INDEX OF MAX STANDARD DEVIATIONS
        time.sleep(0)
        signal_std_breakpoint = synchronized_data[sig_max_std[index]].loc[synchronized_data[sig_counter_list[index]
        #CREATING SECTION OF SIGNALS
        list_points = list(range(1, (len(signal_std_breakpoint))))
        for start_point in list_points:
            if start_point+1 in list_points:
                synchronized_data[sig_counter_list[index]].iloc[signal_std_breakpoint[start_point]: signal_std_brea
                time.sleep(0)
            # IF THERE IS ANY SIGNAL AFTER THIS SIGNAL THAT NEEDED TO BE RENUMBERED
            if start_point == list_points[-1] and synchronized_data[sig_counter_list[index]].iloc[signal_std_breakp
                synchronized_data[sig_counter_list[index]].iloc[signal_std_breakpoint[-1]+1:].loc[synchronized_data
```

```
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
############################################################################ DETECTING PARTS OF A SIGNAL USING STD #############
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################

synchronized_data = synchronized_data.set_index('scenario time')

# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#                                                                    TRIM ELEV
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


sig_counter_list =['signal_counter_ailrn', 'signal_counter_elev', 'signal_counter_ruddr']
sig_pilot = ['signal_ailron_pilot', 'signal_elev_pilot', 'signal_ruddr_pilot']

signal_ailrn_onset_timeindex = []
signal_elev_onset_timeindex = []
signal_ruddr_onset_timeindex = []

for index, member in enumerate(sig_counter_list):
    synchronized_data[sig_pilot[index]] = np.nan
    signal_onset_index = synchronized_data[member].unique()

    if index == 0:
        signal_ailrn_onset_index = [x for x in signal_onset_index if ~np.isnan(x)]
        for i in signal_ailrn_onset_index:
            signal_ailrn_onset_timeindex.append(synchronized_data[member].loc[synchronized_data[member] == i].index[0])
            synchronized_data[sig_pilot[index]][synchronized_data[member].loc[synchronized_data[member] == i].index[0]]

        sp_ailrn = [np.timedelta64(signal_ailrn_onset_timeindex[x], 'ns') / pd.Timedelta(1,'ns') for x in range(len(sig

    elif index == 1:
        signal_elev_onset_index = [x for x in signal_onset_index if ~np.isnan(x)]
        for i in signal_elev_onset_index:
```

```
            signal_elev_onset_timeindex.append(synchronized_data[member].loc[synchronized_data[member] == i].index[0])
            synchronized_data[sig_pilot[index]][synchronized_data[member].loc[synchronized_data[member] == i].index[0]]

        sp_elev = [np.timedelta64(signal_elev_onset_timeindex[x], 'ns') / pd.Timedelta(1,'ns') for x in range(len(signa

    else:
        signal_ruddr_onset_index = [x for x in signal_onset_index if ~np.isnan(x)]
        for i in signal_ruddr_onset_index:
            signal_ruddr_onset_timeindex.append(synchronized_data[member].loc[synchronized_data[member] == i].index[0])
            synchronized_data[sig_pilot[index]][synchronized_data[member].loc[synchronized_data[member] == i].index[0]]
        sp_ruddr = [np.timedelta64(signal_ruddr_onset_timeindex[x], 'ns') / pd.Timedelta(1,'ns') for x in range(len(sig

# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#                                                                 TRIM ELEV
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

style.available
style.use('classic')

plt.rcParams.update({'font.size': 9})
plt.rcParams["font.weight"] = "bold"
plt.rcParams["axes.labelweight"] = "bold"

fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(20, 10), sharex=True)

axes[0].set_xlim([0, np.timedelta64(synchronized_data.index[-1], 'ns') / pd.Timedelta(1,'ns')])

for i in range(3):
    axes[i].xaxis.grid(True)
    axes[i].minorticks_on()
    axes[i].xaxis.grid(True, which="both")
    axes[i].grid(b=True, axis='both', which='minor', linestyle=':', linewidth='0.5', color='black')
    axes[i].grid(b=True, axis='both', which='major', linestyle='--', linewidth='0.5', color='black')
    axes[i].set_ylim([-1.015, 1.015])

trim_elev_timeindex = []
for i in synchronized_data['singal_trim, elev_pct_change'].unique():
```

```
    if i>0:
        trim_elev_timeindex.append(synchronized_data['singal_trim, elev_pct_change'].loc[synchronized_data['singal_trim

sp_trim_elev = [np.timedelta64(trim_elev_timeindex[x], 'ns') / pd.Timedelta(1,'ns') for x in range(len(trim_elev_timein

# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# TRIM ELEV
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
synchronized_data['signal_elev_detect'].plot(ax=axes[0], linestyle= '-', drawstyle='steps', linewidth = 3, c='g', zorde
synchronized_data['elev, yoke1'].plot(ax=axes[0], linestyle= '-' , linewidth = 2, style='b-', marker='+', c='b', ms = 4
synchronized_data['elev, yoke1'].rolling(window).agg([agg]).plot(ax=axes[0], linestyle= '-' , linewidth = 2, style='b-'
synchronized_data['signal_elev_max_std'].plot(ax=axes[0], style='.', marker = '*', c='black', ms= 8, zorder=3)

synchronized_data['signal_ailrn_detect'].plot(ax=axes[1], linestyle= '-', drawstyle='steps', linewidth = 3, c='g', zord
synchronized_data['signal_ailrn_max_std'].plot(ax=axes[1], style='.', marker = '*', c='black', ms= 8, zorder=3)
synchronized_data['ailrn, yoke1'].plot(ax=axes[1], linestyle= '-' , linewidth = 2, style='b-', marker='+', c='b', ms =
synchronized_data['ailrn, yoke1'].rolling(window).agg([agg]).plot(ax=axes[1], linestyle= '-' , linewidth = 2, style='b-

synchronized_data['signal_ruddr_detect'].plot(ax=axes[2], linestyle= '-', drawstyle='steps', linewidth = 3, c='g', zord
synchronized_data['signal_ruddr_max_std'].plot(ax=axes[2], style='.', marker = '*', c='black', ms= 8, zorder=3)
synchronized_data['ruddr, yoke1'].plot(ax=axes[2], linestyle= '-' , linewidth = 2, style='b-', marker='+', c='b', ms =
synchronized_data['ruddr, yoke1'].rolling(window).agg([agg]).plot(ax=axes[2], linestyle= '-' , linewidth = 2, style='b-

# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# TRIM ELEV
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# START POINTS OF ELEV SIGNALS
for i in sp_elev:
    axes[0].axvline(x = i , color='teal', linestyle='dashed', linewidth=2.5)

for i in sp_trim_elev:
    axes[0].axvline(x = i , color='blue', linestyle='dashed' , linewidth=2.5)
```

```
for i in sp_ailrn:
    axes[1].axvline(x = i , color='teal', linestyle='dashed' , linewidth=2.5)

for i in sp_ruddr:
    axes[2].axvline(x = i , color='teal', linestyle='dashed' , linewidth=2.5)

for i in range(3):
    axes[i].legend(loc='lower right', framealpha = 0.6)
    axes[i].grid()

plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)

plt.savefig('_'  + file_name + '_' + task_name + '_'  + 'segmentation_signal.png', bbox_inches='tight')
plt.savefig('_'  + file_name + '_' + task_name + '_'  + 'segmentation_signal.pdf', bbox_inches='tight')

############################################################              3rd CHART                 ###############
############################################################              3rd CHART                 ###############
############################################################              3rd CHART                 ###############
############################################################              3rd CHART                 ###############
############################################################              3rd CHART                 ###############


plt.show()

#########################################################################################################################
#########################################################################################################################
#########################################################################################################################
########################################################################## AGGREGATING ALL DATA IN A DATAFRAME AND SAVING IT #
#########################################################################################################################
#########################################################################################################################
#########################################################################################################################
#########################################################################################################################
#########################################################################################################################
#########################################################################################################################
##########################################################################         DETECTING RESPONSE      ###################
#########################################################################################################################
#########################################################################################################################
#########################################################################################################################
# CREATES NUMBERS FOR EACH AOIs. THEN I WILL USE THIS TO ASSESS THE CURRENT AOI AND PREVIOUS AOI
#########################################################################################################################
#########################################################################################################################
```

```
####################################################################################################################################
##################################################################         COUNTING NUMBER OF AOIs         ###########
####################################################################################################################################
####################################################################################################################################
####################################################################################################################################
index_aoi = []
index_aoi = synchronized_data['fixation'].loc[synchronized_data['fixation']==2].index.tolist()
index_aoi.append(synchronized_data.index[-1])

synchronized_data['aoi_counter'] = np.nan
synchronized_data['response_elev'] = np.nan
synchronized_data['response_ailrn'] = np.nan
synchronized_data['response_elev_trim'] = np.nan

aoi_counter = 1

# IT CREATES A COLUMN AND KEEP THE NUMBER OF AOI IN ORDER TO BE CHECKED AFTER JUST FOR FIXATIONS NOT SACCADE

response_flag = False

for counter in range(len(index_aoi)-1):
    signal_detected_elev_timeindex = []
    signal_detected_elev_trim_timeindex = []
    signal_detected_ailron_timeindex = []
    signal_detected_ruddr_timeindex = []

    synchronized_data['aoi_counter'][index_aoi[counter]:index_aoi[counter+1]] = aoi_counter
    aoi_counter += 1
####################################################################################################################################
####################################################################################################################################
####################################################################################################################################
##################################################################         COUNTING NUMBER OF AOIs         ###########
####################################################################################################################################
####################################################################################################################################
####################################################################################################################################

    ################################################################################################################################
    ################################################################################################################################
    ################################################################################################################################
    ##################################################################                          RESPONSE TO ALTITUDE
    ##################################################################                          RESPONSE TO ALTITUDE
```

```
        ######################################################################                    RESPONSE TO ALTITUDE
        ######################################################################                    RESPONSE TO ALTITUDE
        ######################################################################                    RESPONSE TO ALTITUDE
        ###########################################################################################################################
        ###########################################################################################################################
        ###########################################################################################################################
# AREA PLOT
# https://python-graph-gallery.com/area-plot/

  # HERE IS THE IDEA, PILOTS CONTROLS THE AIRPLANE'S ALTITUDE BY LOOKING AT AI / ALT / AND TRIM
  # THE CODE HAS TWO PART : FIRST IT ASSESSES THE AI / ALT AND THEN TRIM INPUTS
  # THIS IS COMMON FOR LEVEL FLIGHT AND LEVEL TURN

  if task_name in {'Level_Flight', 'Level_Turn'}:
      response_flag = True


      for time_index in signal_elev_onset_timeindex:
          # INDEX NUMBER OF TIME BASED INDEX DATAFRAME
          row_time_index = synchronized_data['Eye movement type'].index.get_loc(time_index)
          # SOMETIMES A RESPONSE FALLS ON A FIXATION AND
          if synchronized_data['Eye movement type'].iloc[row_time_index] == 'Fixation':
              if synchronized_data['aoi_counter'].iloc[row_time_index] > 1:
                  if (synchronized_data['AOI'][row_time_index] == 'ALT') or \
                  (synchronized_data['AOI'][row_time_index] == 'AI' and synchronized_data['AOI'].loc[synchronized_data['a
                      synchronized_data['response_elev'].iloc[row_time_index] = 1
                      signal_detected_elev_timeindex.append(time_index)
                  else:
                      time.sleep(0.0)

              elif synchronized_data['aoi_counter'].iloc[row_time_index] == 1:
                  if (synchronized_data['AOI'][row_time_index] in {'ALT', 'AI'}):
                      synchronized_data['response_elev'].iloc[row_time_index] = 1
                      signal_detected_elev_timeindex.append(time_index)
                  else:
                      time.sleep(0.0)
          # SOMETIMES A RESPONSE FALLS ON A SACCADE
          elif synchronized_data['Eye movement type'].iloc[row_time_index] == 'Saccade':
              if synchronized_data['aoi_counter'].iloc[row_time_index] > 1:
                  if (synchronized_data['AOI'].iloc[row_time_index+1] == 'ALT') or \
                  (synchronized_data['AOI'].iloc[row_time_index+1] == 'AI' and synchronized_data['AOI'].loc[synchronized_
```

```
                    synchronized_data['response_elev'].iloc[row_time_index] = 1
                    signal_detected_elev_timeindex.append(time_index)
                else:
                    time.sleep(0.0)

            elif synchronized_data['aoi_counter'].iloc[row_time_index] == 1:
                if (synchronized_data['AOI'].iloc[row_time_index+1] in {'ALT', 'AI'}):
                    synchronized_data['response_elev'].iloc[row_time_index] = 1
                    signal_detected_elev_timeindex.append(time_index)
                else:
                    time.sleep(0.0)

    for time_index in trim_elev_timeindex:
        # INDEX NUMBER OF TIME BASED INDEX DATAFRAME
        row_time_index = synchronized_data['Eye movement type'].index.get_loc(time_index)
        # SOMETIMES A RESPONSE FALLS ON A FIXATION AND
        if synchronized_data['Eye movement type'].iloc[row_time_index] == 'Fixation':
            if (synchronized_data['AOI'][row_time_index] in {'ALT', 'AI'}):
                synchronized_data['response_elev_trim'].iloc[row_time_index] = 1
                signal_detected_elev_trim_timeindex.append(time_index)
            else:
                time.sleep(0.0)
        # SOMETIMES A RESPONSE FALLS ON A SACCADE
        elif synchronized_data['Eye movement type'].iloc[row_time_index] == 'Saccade':
            if (synchronized_data['AOI'].iloc[row_time_index+1] in {'ALT', 'AI'}):
                synchronized_data['response_elev_trim'].iloc[row_time_index] = 1
                signal_detected_elev_trim_timeindex.append(time_index)
            else:
                time.sleep(0.0)

elif task_name in {'Descent'}:
    response_flag = True

    for time_index in signal_elev_onset_timeindex:
        # INDEX NUMBER OF TIME BASED INDEX DATAFRAME
        row_time_index = synchronized_data['Eye movement type'].index.get_loc(time_index)
        # SOMETIMES A RESPONSE FALLS ON A FIXATION AND
        if synchronized_data['Eye movement type'].iloc[row_time_index] == 'Fixation':
            if synchronized_data['aoi_counter'].iloc[row_time_index] > 1:
                if (synchronized_data['AOI'][row_time_index] in {'VSI', 'ASI', 'ALT'}) or \
                (synchronized_data['AOI'][row_time_index] == 'AI' and (synchronized_data['AOI'].loc[synchronized_data['
```

```
                    synchronized_data['response_elev'].iloc[row_time_index] = 1
                    signal_detected_elev_timeindex.append(time_index)
                else:
                    time.sleep(0.0)

            elif synchronized_data['aoi_counter'].iloc[row_time_index] == 1:
                if (synchronized_data['AOI'][row_time_index] in {'VSI', 'ASI', 'ALT', 'AI'}):
                    synchronized_data['response_elev'].iloc[row_time_index] = 1
                    signal_detected_elev_timeindex.append(time_index)
                else:
                    time.sleep(0.0)


    for time_index in trim_elev_timeindex:
        # INDEX NUMBER OF TIME BASED INDEX DATAFRAME
        row_time_index = synchronized_data['Eye movement type'].index.get_loc(time_index)
        # SOMETIMES A RESPONSE FALLS ON A FIXATION AND
        if synchronized_data['Eye movement type'].iloc[row_time_index] == 'Fixation':
            if (synchronized_data['AOI'][row_time_index] in {'VSI', 'ASI', 'ALT', 'AI'}):
                synchronized_data['response_elev_trim'].iloc[row_time_index] = 1
                signal_detected_elev_trim_timeindex.append(time_index)
            else:
                time.sleep(0.0)
        # SOMETIMES A RESPONSE FALLS ON A SACCADE
        elif synchronized_data['Eye movement type'].iloc[row_time_index] == 'Saccade':
            if (synchronized_data['AOI'].iloc[row_time_index+1] in {'VSI', 'ASI', 'ALT', 'AI'}):
                synchronized_data['response_elev_trim'].iloc[row_time_index] = 1
                signal_detected_elev_trim_timeindex.append(time_index)
            else:
                time.sleep(0.0)


elif task_name in {'Landing'}:
    response_flag = True

    for time_index in signal_elev_onset_timeindex:
        # INDEX NUMBER OF TIME BASED INDEX DATAFRAME
        row_time_index = synchronized_data['Eye movement type'].index.get_loc(time_index)
        # SOMETIMES A RESPONSE FALLS ON A FIXATION AND
        if synchronized_data['Eye movement type'].iloc[row_time_index] == 'Fixation':
            if synchronized_data['aoi_counter'].iloc[row_time_index] > 1:
```

```
                if (synchronized_data['AOI'][row_time_index] in {'OSW', 'VSI', 'ASI', 'ALT'}) or \
                (synchronized_data['AOI'][row_time_index] == 'AI' and (synchronized_data['AOI'].loc[synchronized_data['
                    synchronized_data['response_elev'].iloc[row_time_index] = 1
                    signal_detected_elev_timeindex.append(time_index)
                else:
                    time.sleep(0.0)

            elif synchronized_data['aoi_counter'].iloc[row_time_index] == 1:
                if (synchronized_data['AOI'][row_time_index] in {'OSW', 'VSI', 'ASI', 'ALT', 'AI'}):
                    synchronized_data['response_elev'].iloc[row_time_index] = 1
                    signal_detected_elev_timeindex.append(time_index)
                else:
                    time.sleep(0.0)

    for time_index in trim_elev_timeindex:
        # INDEX NUMBER OF TIME BASED INDEX DATAFRAME
        row_time_index = synchronized_data['Eye movement type'].index.get_loc(time_index)
        # SOMETIMES A RESPONSE FALLS ON A FIXATION AND
        if synchronized_data['Eye movement type'].iloc[row_time_index] == 'Fixation':
            if (synchronized_data['AOI'][row_time_index] in {'OSW', 'VSI', 'ASI', 'ALT', 'AI'}):
                synchronized_data['response_elev_trim'].iloc[row_time_index] = 1
                signal_detected_elev_trim_timeindex.append(time_index)
            else:
                time.sleep(0.0)
        # SOMETIMES A RESPONSE FALLS ON A SACCADE
        elif synchronized_data['Eye movement type'].iloc[row_time_index] == 'Saccade':
            if (synchronized_data['AOI'].iloc[row_time_index+1] in {'OSW', 'VSI', 'ASI', 'ALT', 'AI'}):
                synchronized_data['response_elev_trim'].iloc[row_time_index] = 1
                signal_detected_elev_trim_timeindex.append(time_index)
            else:
                time.sleep(0.0)


#######################################################################################################################
#######################################################################################################################
#######################################################################################################################
###################################################################                           RESPONSE TO HEADING
###################################################################                           RESPONSE TO HEADING
###################################################################                           RESPONSE TO HEADING
###################################################################                           RESPONSE TO HEADING
###################################################################                           RESPONSE TO HEADING
```

```
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
if task_name in {'Level_Flight', 'Level_Turn', 'Descent'}:
    response_flag = True

    for time_index in signal_ailrn_onset_timeindex:
        # INDEX NUMBER OF TIME BASED INDEX DATAFRAME
        row_time_index = synchronized_data['Eye movement type'].index.get_loc(time_index)
        # GETS INDEX OF TIME INDEX DATAFRAME
        # SOMETIMES A RESPONSE FALLS ON A FIXATION AND
        if synchronized_data['Eye movement type'].iloc[row_time_index] == 'Fixation':
            if synchronized_data['aoi_counter'].iloc[row_time_index] > 1:
                if (synchronized_data['AOI'][row_time_index] in {'HI', 'TC'}) or (synchronized_data['AOI'][row_time_ind
                    synchronized_data['response_ailrn'].iloc[row_time_index] = 1
                    signal_detected_ailron_timeindex.append(time_index)
                else:
                    time.sleep(0.0)
            elif synchronized_data['aoi_counter'].iloc[row_time_index] == 1:
                if (synchronized_data['AOI'][row_time_index] in {'HI', 'TC', 'AI'}):
                    synchronized_data['response_ailrn'].iloc[row_time_index] = 1
                    signal_detected_ailron_timeindex.append(time_index)
                else:
                    time.sleep(0.0)
        # SOMETIMES A RESPONSE FALLS ON A SACCADE
        elif synchronized_data['Eye movement type'].iloc[row_time_index] == 'Saccade':
            if synchronized_data['aoi_counter'].iloc[row_time_index] > 1:
                if (synchronized_data['AOI'].iloc[row_time_index+1] in {'HI', 'TC'}) or (synchronized_data['AOI'].iloc[
                    synchronized_data['response_ailrn'].iloc[row_time_index] = 1
                    signal_detected_ailron_timeindex.append(time_index)
                else:
                        time.sleep(0.0)
            elif synchronized_data['aoi_counter'].iloc[row_time_index] == 1:
                if (synchronized_data['AOI'][row_time_index] in { 'HI', 'TC' , 'AI'}):
                    synchronized_data['response_ailrn'].iloc[row_time_index] = 1
                    signal_detected_ailron_timeindex.append(time_index)
            else:
                time.sleep(0.0)


elif task_name == 'Landing':
```

```
        response_flag = True
        for time_index in signal_ailrn_onset_timeindex:
            # INDEX NUMBER OF TIME BASED INDEX DATAFRAME
            row_time_index = synchronized_data['Eye movement type'].index.get_loc(time_index)
            # GETS INDEX OF TIME INDEX DATAFRAME
            # SOMETIMES A RESPONSE FALLS ON A FIXATION AND
            if synchronized_data['Eye movement type'].iloc[row_time_index] == 'Fixation':
                if synchronized_data['aoi_counter'].iloc[row_time_index] > 1:
                    if (synchronized_data['AOI'][row_time_index] in {'OSW', 'HI', 'TC'}) or (synchronized_data['AOI'][row_t
                        synchronized_data['response_ailrn'].iloc[row_time_index] = 1
                        signal_detected_ailron_timeindex.append(time_index)
                    else:
                        time.sleep(0.0)
                elif synchronized_data['aoi_counter'].iloc[row_time_index] == 1:
                    if (synchronized_data['AOI'][row_time_index] in {'OSW', 'HI', 'TC', 'AI'}):
                        synchronized_data['response_ailrn'].iloc[row_time_index] = 1
                        signal_detected_ailron_timeindex.append(time_index)
                    else:
                        time.sleep(0.0)
            # SOMETIMES A RESPONSE FALLS ON A SACCADE
            elif synchronized_data['Eye movement type'].iloc[row_time_index] == 'Saccade':
                if synchronized_data['aoi_counter'].iloc[row_time_index] > 1:
                    if (synchronized_data['AOI'].iloc[row_time_index+1] in {'OSW', 'HI', 'TC'}) or (synchronized_data['AOI'
                        synchronized_data['response_ailrn'].iloc[row_time_index] = 1
                        signal_detected_ailron_timeindex.append(time_index)
                    else:
                            time.sleep(0.0)
                elif synchronized_data['aoi_counter'].iloc[row_time_index] == 1:
                    if (synchronized_data['AOI'][row_time_index] in {'OSW', 'HI', 'TC', 'AI'}):
                        synchronized_data['response_ailrn'].iloc[row_time_index] = 1
                        signal_detected_ailron_timeindex.append(time_index)
                else:
                    time.sleep(0.0)


    if task_name in {'Level_Flight', 'Level_Turn', 'Descent'}:
        response_flag = True

        for time_index in signal_ruddr_onset_timeindex:
            # INDEX NUMBER OF TIME BASED INDEX DATAFRAME
            row_time_index = synchronized_data['Eye movement type'].index.get_loc(time_index)
```

```
            # GETS INDEX OF TIME INDEX DATAFRAME
            # SOMETIMES A RESPONSE FALLS ON A FIXATION AND
            if synchronized_data['Eye movement type'].iloc[row_time_index] == 'Fixation':
                if synchronized_data['aoi_counter'].iloc[row_time_index] >= 1:
                    if (synchronized_data['AOI'][row_time_index] in {'TC'}):
                        synchronized_data['response_ailrn'].iloc[row_time_index] = 1
                        signal_detected_ruddr_timeindex.append(time_index)
                    else:
                        time.sleep(0.0)

            # SOMETIMES A RESPONSE FALLS ON A SACCADE
            elif synchronized_data['Eye movement type'].iloc[row_time_index] == 'Saccade':
                if synchronized_data['aoi_counter'].iloc[row_time_index] >= 1:
                    if (synchronized_data['AOI'].iloc[row_time_index+1] in {'TC'}):
                        synchronized_data['response_ailrn'].iloc[row_time_index] = 1
                        signal_detected_ruddr_timeindex.append(time_index)
                    else:
                            time.sleep(0.0)

    elif task_name == 'Landing':
        response_flag = True
        for time_index in signal_ruddr_onset_timeindex:
            # INDEX NUMBER OF TIME BASED INDEX DATAFRAME
            row_time_index = synchronized_data['Eye movement type'].index.get_loc(time_index)
            # GETS INDEX OF TIME INDEX DATAFRAME
            # SOMETIMES A RESPONSE FALLS ON A FIXATION AND
            if synchronized_data['Eye movement type'].iloc[row_time_index] == 'Fixation':
                if synchronized_data['aoi_counter'].iloc[row_time_index] >= 1:
                    if (synchronized_data['AOI'][row_time_index] in {'TC', 'OSW'}):
                        synchronized_data['response_ailrn'].iloc[row_time_index] = 1
                        signal_detected_ruddr_timeindex.append(time_index)
                    else:
                        time.sleep(0.0)

            # SOMETIMES A RESPONSE FALLS ON A SACCADE
            elif synchronized_data['Eye movement type'].iloc[row_time_index] == 'Saccade':
                if synchronized_data['aoi_counter'].iloc[row_time_index] >= 1:
                    if (synchronized_data['AOI'].iloc[row_time_index+1] in {'TC', 'OSW'}):
                        synchronized_data['response_ailrn'].iloc[row_time_index] = 1
                        signal_detected_ruddr_timeindex.append(time_index)
                    else:
```

```
                    time.sleep(0.0)

##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
#########################################################################          AILRON ==> CHECKING IF SIGNAL IS A RESPONSE
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
#########################################################################          AILRON ==> CHECKING IF SIGNAL IS A RESPONSE
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################
#########################################################################                                DETECTING RESPONSE
##########################################################################################################################
##########################################################################################################################
##########################################################################################################################

style.available
style.use('classic')

plt.rcParams.update({'font.size': 9})
plt.rcParams["font.weight"] = "bold"
plt.rcParams["axes.labelweight"] = "bold"

fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(20, 10), sharex=True)

axes[0].set_xlim([0, np.timedelta64(synchronized_data.index[-1], 'ns') / pd.Timedelta(1,'ns')])

for i in range(3):
    axes[i].xaxis.grid(True)
    axes[i].minorticks_on()
    axes[i].xaxis.grid(True, which="both")
    axes[i].grid(b=True, axis='both', which='minor', linestyle=':', linewidth='0.5', color='black')
    axes[i].grid(b=True, axis='both', which='major', linestyle='--', linewidth='0.5', color='black')
```

```
    axes[i].set_ylim([-1.015, 1.015])
# THE METHOD FOR FIDING THE ONSET OF ELEV SIGNALS: (1) FINDING UNIQUE ITEMS OF signal_counter_elev (2) REMOVING NAN FRO
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

synchronized_data['signal_elev_detect'].plot(ax=axes[0], linestyle= '-', drawstyle='steps', linewidth = 3, c='g', zorde
synchronized_data['signal_ailrn_detect'].plot(ax=axes[1], linestyle= '-', drawstyle='steps', linewidth = 3, c='g', zord
synchronized_data['signal_ruddr_detect'].plot(ax=axes[2], linestyle= '-', drawstyle='steps', linewidth = 3, c='g', zord

synchronized_data['elev, yoke1'].plot(ax=axes[0], linestyle= '-' , linewidth = 2, style='b-', marker='+', c='b', ms = 4
synchronized_data['elev, yoke1'].rolling(window).agg([agg]).plot(ax=axes[0], linestyle= '-' , linewidth = 2, style='b-'
synchronized_data['signal_elev_max_std'].plot(ax=axes[0], style='.', marker = '*', c='black', ms= 8, zorder=3)

synchronized_data['ailrn, yoke1'].plot(ax=axes[1], linestyle= '-' , linewidth = 2, style='b-', marker='+', c='b', ms =
synchronized_data['ailrn, yoke1'].rolling(window).agg([agg]).plot(ax=axes[1], linestyle= '-' , linewidth = 2, style='b-
synchronized_data['signal_ailrn_max_std'].plot(ax=axes[1], style='.', marker = '*', c='black', ms= 8, zorder=3)

synchronized_data['ruddr, yoke1'].plot(ax=axes[2], linestyle= '-' , linewidth = 2, style='b-', marker='+', c='b', ms =
synchronized_data['ruddr, yoke1'].rolling(window).agg([agg]).plot(ax=axes[2], linestyle= '-' , linewidth = 2, style='b-
synchronized_data['signal_ruddr_max_std'].plot(ax=axes[2], style='.', marker = '*', c='black', ms= 8, zorder=3)


if response_flag == True:
    sp_response_detect_elev = [np.timedelta64(signal_detected_elev_timeindex[x], 'ns') / pd.Timedelta(1,'ns') for x in
    sp_response_detect_elev_trim = [np.timedelta64(signal_detected_elev_timeindex[x], 'ns') / pd.Timedelta(1,'ns') for
    sp_response_detect_ailron = [np.timedelta64(signal_detected_ailron_timeindex[x], 'ns') / pd.Timedelta(1,'ns') for x

    for i in sp_response_detect_elev:
        axes[0].axvline(x = i , color='orange', linestyle='dashed', linewidth=2.5, zorder = 6)

    for i in sp_response_detect_elev_trim:
        axes[0].axvline(x = i , color='orange', linestyle='dashed', linewidth=2.5, zorder = 6)

    for i in sp_response_detect_ailron:
        axes[1].axvline(x = i , color='orange', linestyle='dashed', linewidth=2.5, zorder = 6)

for i in sp_trim_elev:
    axes[0].axvline(x = i , color='blue', linestyle='dashed', linewidth=2.5, zorder = 4)

for i in sp_elev:
    axes[0].axvline(x = i , color='teal', linestyle='dashed', linewidth=2.5)
```

```
            for i in sp_ailrn:
                axes[1].axvline(x = i , color='teal', linestyle='dashed' , linewidth=2.5)

            for i in sp_ruddr:
                axes[2].axvline(x = i , color='teal', linestyle='dashed' , linewidth=2.5)

            for i in range(3):
                axes[i].legend(loc='lower right', framealpha = 0.6)
                axes[i].grid()

            plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
            plt.savefig('_'  + file_name + '_' + task_name + '_'  + 'segmentation_signal_detected.png', bbox_inches='tight')
            plt.savefig('_'  + file_name + '_' + task_name + '_'  + 'segmentation_signal_detected.pdf', bbox_inches='tight')
            plt.show()

            # AREA UNDER THE CURVE FOR SIGNALS OF ELEV / AILRON / RUDDER
            # IT IS CALCULATED BASED ON ABSOLUTE VALUE OF CURVES

            # https://stackoverflow.com/questions/31964727/integrating-discrete-point-in-python
            # https://stackoverflow.com/questions/44915116/how-to-decide-between-scipy-integrate-simps-or-numpy-trapz

            y_elev_points = synchronized_data['elev, yoke1'].abs().as_matrix()
            y_elev_points_norm = np.full_like(y_elev_points, 1, dtype=np.double)
            x_elev_points = np.arange(synchronized_data['elev, yoke1'].shape[0])

            y_ailrn_points = synchronized_data['ailrn, yoke1'].abs().as_matrix()
            y_ailrn_points_norm = np.full_like(y_ailrn_points, 1, dtype=np.double)
            x_ailrn_points = np.arange(synchronized_data['ailrn, yoke1'].shape[0])

            y_ruddr_points = synchronized_data['ruddr, yoke1'].abs().as_matrix()
            y_ruddr_points_norm = np.full_like(y_ruddr_points, 1, dtype=np.double)
            x__ruddr_points = np.arange(synchronized_data['ruddr, yoke1'].shape[0])

            synchronized_data['area_elev'] = scipy.integrate.simps(x=x_elev_points, y= y_elev_points) / scipy.integrate.simps(x=x_e
            synchronized_data['area_ailrn'] = scipy.integrate.simps(x=x_ailrn_points, y= y_ailrn_points) / scipy.integrate.simps(x=
            synchronized_data['area_ruddr'] = scipy.integrate.simps(x=x__ruddr_points, y= y_ruddr_points) / scipy.integrate.simps(x

#             #%%
#             synchronized_data['deviation altitude'][signal_elev_onset_timeindex[1]]
#             #%%
#             for points in signal_elev_onset_timeindex:
```

```
#                print(synchronized_data['deviation heading'][points])

    ############################################################                4th CHART                ###############
    ############################################################                4th CHART                ###############
    ############################################################                4th CHART                ###############
    ############################################################                4th CHART                ###############
    ############################################################                4th CHART                ###############

    synchronized_data['cal_time'] = np.nan

    for index, row in synchronized_data.iterrows():

        row_index  = synchronized_data.index.get_loc(index)

        if task_name == 'Level_Flight':

            if synchronized_data.index.get_loc(index)+50 <= synchronized_data.index.get_loc(synchronized_data.index[-1]):
                if (synchronized_data['deviation heading'][synchronized_data.index.get_loc(index):synchronized_data.index.g
                (abs(synchronized_data['deviation altitude'][synchronized_data.index.get_loc(index):synchronized_data.index
                    print(row_index, 'NO DEVIATION:  ', np.timedelta64(index, 's') / pd.Timedelta(1,'s'))
                    synchronized_data['cal_time'] = np.timedelta64(index, 's') / pd.Timedelta(1,'s')
                    break
        elif task_name == 'Level_Turn':
            if synchronized_data['Training'].iloc[row_index] == 'Before Treatment':
                if 207.5 <= synchronized_data['heading mag'].iloc[row_index] <= 212.5:
                    synchronized_data['cal_time'] = np.timedelta64(index, 's') / pd.Timedelta(1,'s')

            elif synchronized_data['Training'].iloc[row_index] == 'After Treatment':
                if 87.5 <= synchronized_data['heading mag'].iloc[row_index] <= 92.5:
                    synchronized_data['cal_time'] = np.timedelta64(index, 's') / pd.Timedelta(1,'s')

        elif task_name == 'Descent':
            if synchronized_data['Training'].iloc[row_index] == 'Before Treatment':
                if 2495 <= synchronized_data['alt, ind'].iloc[row_index] <= 2505:
                    synchronized_data['cal_time'] = np.timedelta64(index, 's') / pd.Timedelta(1,'s')

            if synchronized_data['Training'].iloc[row_index] == 'After Treatment':
                if 2005 <= synchronized_data['alt, ind'].iloc[row_index] <= 1995:
                    synchronized_data['cal_time'] = np.timedelta64(index, 's') / pd.Timedelta(1,'s')
```

```
            elif task_name == 'Landing':
                synchronized_data['cal_time'] = np.timedelta64(synchronized_data['landed or not'].loc[synchronized_data['landed


        if type(synchronized_data.index) == pd.core.indexes.timedeltas.TimedeltaIndex:
#             print('STEP 1: INDEX IS A RANGE AND IT SHOULD BE')
            time.sleep(0.0)
        else:
#             print('STEP 2')
            synchronized_data = synchronized_data.set_index('scenario time')

        # Eventplot
        #https://matplotlib.org/gallery/lines_bars_and_markers/eventplot_demo.html#sphx-glr-gallery-lines-bars-and-markers-even

        # signal
        #https://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtime-timeseries-data/43512887#43512887

        # https://www.quora.com/Simple-algorithm-for-trend-detection-in-time-series-data
#             #%%
#             synchronized_data.index[-1] - synchronized_data.index[0]

#             eye_data['scenario time'].loc[eye_data['scenario time'] < '00:01:00']

        # Bollinger Band
        formatter = mticker.ScalarFormatter(useMathText=True)
        formatter.set_powerlimits((-3,2))

        style.available
        #style.use('ggplot')
        style.use('classic') #sets the size of the charts

        plt.rcParams.update({'font.size': 12})
        plt.rcParams["font.weight"] = "bold"
        plt.rcParams["axes.labelweight"] = "bold"

        x = 0
        while (x < len(list_interval)-1 and (synchronized_data[list_interval[x]:list_interval[x+1]]).empty == False):

            fig, axes = plt.subplots(nrows=8, ncols=1, figsize=(14, 20), sharex=True)

            for plot_index in {0,1,2,3,4,5,6,7}:
```

```
    axes[plot_index].minorticks_on()
    axes[plot_index].grid(axis='x', which='minor', linestyle=':', linewidth='0.5', color='black')
    axes[plot_index].set_yticks(range(0,2))
    axes[plot_index].set_ylim([-0.025, 1.015])
    axes[plot_index].set_axisbelow(True)
    axes[plot_index].xaxis.grid(True)

for i in {0,1,2,3,4,5,6}:

    axes[i].yaxis.set_major_formatter(formatter)
    axes[i].yaxis.offsetText.set_visible(False)

    if AOI_list[i] == 'AI':
        time.sleep(0.0)

        axes[i+1].set_ylabel('AI')

        ax3 = axes[i+1].twinx()

        rspine = ax3.spines['right']
        rspine.set_position(('axes', 1.07))

        ax3.set_frame_on(True)
        ax3.patch.set_visible(False)

        yticks = ticker.MaxNLocator(5)
        ax3.yaxis.set_major_locator(yticks)
        ax3.set_ylabel('Roll, deg.')

        synchronized_data['AI'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronized_data['t
        synchronized_data['pitch, deg'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronized
        synchronized_data['roll, deg'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronized_

        plt.legend([axes[i+1].get_lines()[0], axes[i+1].right_ax.get_lines()[0], ax3.get_lines()[0]], ['AI' , 'Pitc
        plt.ylabel('Pitch, deg.')

        plt.setp(axes[i+1].get_yticklabels(), visible=False)

        ax3.xaxis.grid(True)
        axes[i+1].set_axisbelow(True)
```

```
        yticks = ticker.MaxNLocator(5)
        axes[i+1].right_ax.yaxis.set_major_locator(yticks)
        axes[i+1].right_ax.set_ylim([-20, 20])
        ax3.set_ylim([-100, 100])
        axes[i+1].grid(which='major', axis='x', linestyle='--')

    elif flight_list[i+1] == 'deviation altitude':

        axes[i+1].set_ylabel('ALT')
        ax4 = axes[i+1].twinx()

        rspine = ax4.spines['right']
        rspine.set_position(('axes', 1.07))

        ax4.set_frame_on(True)
        ax4.patch.set_visible(False)

        yticks = ticker.MaxNLocator(5)
        ax4.yaxis.set_major_locator(yticks)
        ax4.set_ylabel('ELEV Input (yoke)')

        synchronized_data['ALT'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronized_data['
        synchronized_data['elev, yoke1'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronize

        if task_name in {'Level_Flight', 'Level_Turn', 'Descent'}:
            synchronized_data['deviation altitude'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (

        elif task_name in {'Landing'}:
            synchronized_data['alt, ind'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchroniz

        if task_name in {'Level_Flight', 'Level_Turn'}:
            threshold_alt_max = 400
            threshold_alt_min = -400

        elif task_name in {'Descent'}:
            threshold_alt_max = 200
            threshold_alt_min = -600

        elif task_name in {'Landing'}:
            threshold_alt_max = 800
            threshold_alt_min = 0
```

```
if task_name not in {'Landing'}:

    if (synchronized_data['deviation altitude'].loc[(synchronized_data['time bracket'] >= list_interval[x])
        #axes[i+1].right_ax.axhline(y= 200, linestyle = '-.', color='purple')
        alt_text = 'ALT GETS TO ' + str(np.round_(synchronized_data['deviation altitude'].loc[(synchronized
        axes[i+1].text(0.5, 0.10, alt_text, verticalalignment='bottom', horizontalalignment='right', transf

    elif (synchronized_data['deviation altitude'].loc[(synchronized_data['time bracket'] >= list_interval[x
        #axes[i+1].right_ax.axhline(y= -200, linestyle = '-.', color='purple')
        alt_text = 'AIRPLANE GETS TO ' + str(np.round_(synchronized_data['deviation altitude'].loc[(synchro
        axes[i+1].text(0.5, 0.10, alt_text, verticalalignment='bottom', horizontalalignment='center', trans

else:
    if (synchronized_data['alt, ind'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchr
        #axes[i+1].right_ax.axhline(y= 200, linestyle = '-.', color='purple')
        alt_text = 'ALT GETS TO ' + str(np.round_(synchronized_data['alt, ind'].loc[(synchronized_data['tim
        axes[i+1].text(0.5, 0.10, alt_text, verticalalignment='bottom', horizontalalignment='right', transf

    elif (synchronized_data['alt, ind'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (sync
        #axes[i+1].right_ax.axhline(y= -200, linestyle = '-.', color='purple')
        alt_text = 'AIRPLANE GETS TO ' + str(np.round_(synchronized_data['alt, ind'].loc[(synchronized_data
        axes[i+1].text(0.5, 0.10, alt_text, verticalalignment='bottom', horizontalalignment='center', trans

if response_flag == True:
    points_counter = 0
    for points in signal_elev_onset_timeindex:
        if points in synchronized_data.loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synch
            if points in signal_detected_elev_timeindex:
                axes[i+1].axvline(sp_elev[points_counter], color='orangered', linestyle='--', linewidth=2.5
            else:
                axes[i+1].axvline(sp_elev[points_counter], color='teal', linestyle='--', linewidth=2.5)
        else:
            time.sleep(0.0)
        points_counter +=1

    points_counter = 0
    for points in trim_elev_timeindex:
        if points in synchronized_data.loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synch
            if points in signal_detected_elev_trim_timeindex:
                axes[i+1].axvline(sp_trim_elev[points_counter], color='violet', linestyle='--', linewidth=2
```

```
                    else:
                        axes[i+1].axvline(sp_trim_elev[points_counter], color='cyan', linestyle='--', linewidth=2.5
                else:
                    time.sleep(0.0)
                points_counter +=1

        ax4.axhline(y= 0.015, linestyle = '--', color='gold')
        ax4.axhline(y=-0.015, linestyle = '--', color='gold')

        if task_name in {'Level_Flight', 'Level_Turn', 'Descent'}:
            plt.legend([axes[i+1].get_lines()[0], axes[i+1].right_ax.get_lines()[0], ax4.get_lines()[0]], ['ALT' ,
            plt.ylabel('ALT CHG ft.')

        elif task_name in {'Landing'}:
            plt.legend([axes[i+1].get_lines()[0], axes[i+1].right_ax.get_lines()[0], ax4.get_lines()[0]], ['ALT' ,
            plt.ylabel('ALT ft.')

        plt.setp(axes[i+1].get_yticklabels(), visible=False)

        ax4.xaxis.grid(True)
        axes[i+1].set_axisbelow(True)
        axes[i+1].xaxis.grid(True)
        ax4.set_ylim([-1, 1])

        yticks = ticker.MaxNLocator(5)
        axes[i+1].right_ax.yaxis.set_major_locator(yticks)

        if task_name in {'Level_Flight', 'Level_Turn'}:
            axes[i+1].right_ax.set_ylim([-400, 400])
        elif task_name in {'Landing'}:
            axes[i+1].right_ax.set_ylim([0, 800])
        elif task_name in {'Descent'}:
            axes[i+1].right_ax.set_ylim([-600, 200])

    elif flight_list[i+1] == 'deviation heading':
        time.sleep(0.0)
        axes[i+1].set_ylabel('HI')
        ax5 = axes[i+1].twinx()
        rspine = ax5.spines['right']
        rspine.set_position(('axes', 1.07))
        ax5.set_frame_on(True)
```

```
ax5.patch.set_visible(False)

yticks = ticker.MaxNLocator(5)
ax5.yaxis.set_major_locator(yticks)
ax5.set_ylabel('AIL Input (yoke)')
ax5.set_ylim([-1.0, 1.0])
ax5.xaxis.grid(True)

synchronized_data['HI'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronized_data['t
synchronized_data['ailrn, yoke1'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchroniz
synchronized_data['deviation heading'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synch

plt.setp(axes[i+1].get_yticklabels(), visible=False)

yticks = ticker.MaxNLocator(5)
axes[i+1].right_ax.yaxis.set_major_locator(yticks)
axes[i+1].grid(which='major', axis='x', linestyle='--')

if response_flag == True:
    points_counter = 0
    for points in signal_ailrn_onset_timeindex:
        if points in synchronized_data.loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synch
            if points in signal_detected_ailron_timeindex:
                axes[i+1].axvline(sp_ailrn[points_counter], color='orangered', linestyle='--', linewidth=2.
            else:
                axes[i+1].axvline(sp_ailrn[points_counter], color='teal', linestyle='--', linewidth=2.5)
        else:
            time.sleep(0.0)
        points_counter +=1

ax5.axhline(y= 0.015, linestyle = '--', color='gold')
ax5.axhline(y=-0.015, linestyle = '--', color='gold')

if task_name in {'Level_Flight', 'Level_Turn', 'Descent'}:
    plt.legend([axes[i+1].get_lines()[0], axes[i+1].right_ax.get_lines()[0], ax5.get_lines()[0]], ['HI' , '
    plt.ylabel('ABS HI DEV deg')
    axes[i+1].right_ax.set_ylim([0, 180])
    axes[i+1].right_ax.set_yticks([0, 45, 90, 135, 180])

elif task_name in {'Landing'}:
    plt.legend([axes[i+1].get_lines()[0], axes[i+1].right_ax.get_lines()[0], ax5.get_lines()[0]], ['HI' , '
```

```
            plt.ylabel('DEV frm RYW CL. deg')
            axes[i+1].right_ax.set_ylim([-10, 10])
            axes[i+1].right_ax.set_yticks([-10, -5, 0, 5, 10])

    elif flight_list[i+1] == 'cal rate turn':

        time.sleep(0.0)
        axes[i+1].set_ylabel('TC')
        synchronized_data['TC'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronized_data['t
        synchronized_data['cal rate turn'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchroni
        axes[i+1].right_ax.set_ylim([-12, 12])
        axes[i+1].right_ax.set_yticks([-12, -6, 0, 6, 12])
        plt.ylabel('Rate of Turn')
        plt.legend([axes[i+1].get_lines()[0], axes[i+1].right_ax.get_lines()[0]], [AOI_list[i] , 'Rate of Turn'] ,

        ax6 = axes[i+1].twinx()

        rspine = ax6.spines['right']
        rspine.set_position(('axes', 1.07))

        ax6.set_frame_on(True)
        ax6.patch.set_visible(False)

        yticks = ticker.MaxNLocator(5)
        ax6.yaxis.set_major_locator(yticks)
        ax6.set_ylabel('RUD Input (yoke)')
        ax6.set_ylim([-1.0, 1.0])

        synchronized_data['ruddr, yoke1'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchroniz

        if response_flag == True:
            points_counter = 0
            for points in signal_ruddr_onset_timeindex:
                if points in synchronized_data.loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synch
                    if points in signal_detected_ruddr_timeindex:
                        axes[i+1].axvline(sp_ruddr[points_counter], color='orangered', linestyle='--', linewidth=2.
                    else:
                        axes[i+1].axvline(sp_ruddr[points_counter], color='teal', linestyle='--', linewidth=2.5)
                else:
                    time.sleep(0.0)
                points_counter +=1
```

```
                    plt.legend([axes[i+1].get_lines()[0], axes[i+1].right_ax.get_lines()[0], ax4.get_lines()[0]], ['TC' , 'Rate
                    plt.ylabel('RUD Input (yoke)')

                elif flight_list[i+1] == 'rpm n, engin':
                    axes[i+1].set_ylabel('TAC')
                    synchronized_data['TAC'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronized_data['
                    synchronized_data['rpm n, engin'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchroniz

                    if task_name in {'Level_Flight', 'Level_Turn', 'Descent'}:
                        axes[i+1].right_ax.set_ylim([1600, 2400])
                        axes[i+1].right_ax.set_yticks([1600, 1800, 2000, 2200, 2400])

                    elif task_name in {'Landing'}:
                        axes[i+1].right_ax.set_ylim([0, 2000])
                        axes[i+1].right_ax.set_yticks([0, 500, 1000, 1500, 2000])

                    plt.ylabel('RPM')
                    plt.legend([axes[i+1].get_lines()[0], axes[i+1].right_ax.get_lines()[0]], [AOI_list[i] , 'RPM'] , loc='uppe

                elif flight_list[i+1] == 'Vind, kias':
                    axes[i+1].set_ylabel('ASI')
                    synchronized_data['ASI'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronized_data['
                    synchronized_data['Vind, kias'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronized_

                    if task_name in {'Level_Flight', 'Level_Turn', 'Descent'}:
                        axes[i+1].right_ax.set_ylim([60, 140])
                        axes[i+1].right_ax.set_yticks([60, 80, 100, 120, 140])

                    elif task_name in {'Landing'}:

                        axes[i+1].right_ax.set_ylim([0, 80])
                        axes[i+1].right_ax.set_yticks([0, 20, 40, 60, 80])

#                       ax7 = axes[i+1].twinx()
#                       rspine = ax7.spines['right']
#                       rspine.set_position(('axes', 1.07))
#                       ax7.set_frame_on(True)
#                       ax7.patch.set_visible(False)
#                       yticks = ticker.MaxNLocator(5)
#                       ax7.yaxis.set_major_locator(yticks)
```

```
#                         ax7.set_ylabel('Flaps Input')
#                         ax7.set_ylim([0.0, 1.0])
#                         ax7.xaxis.grid(True)
#                         synchronized_data['flap, handl'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchr

                        plt.ylabel('ASI, Kias.')
                        plt.legend([axes[i+1].get_lines()[0], axes[i+1].right_ax.get_lines()[0]], [AOI_list[i] , 'ASI, Kias.'] , lo

                    elif flight_list[i+1] == 'VVI, fpm':
                        axes[i+1].set_ylabel('VSI')
                        synchronized_data['VSI'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronized_data['
                        synchronized_data['VVI, fpm'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronized_d

                        axes[i+1].right_ax.set_ylim([-1000, 1000])
                        plt.ylabel('VSI, fpm.')
                        plt.legend([axes[i+1].get_lines()[0], axes[i+1].right_ax.get_lines()[0]], [AOI_list[i] , 'VSI, fpm.'] , loc

                for i in {7}:

                    synchronized_data[AOI_list[i]].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (  synchronized_da
                    synchronized_data[AOI_list[i+1]].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronized_da
                    synchronized_data[AOI_list[i+2]].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronized_da
                    synchronized_data[AOI_list[i+3]].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (synchronized_da

                    plt.setp(axes[i-7].get_yticklabels(), visible=False)

                    lines = axes[i-7].get_lines()
                    axes[i-7].legend(lines, [l.get_label() for l in lines], loc='upper left', framealpha = 0.7, fontsize = 'medium'
                    axes[i-7].set_axisbelow(True)
                    axes[i-7].xaxis.grid(True)
                    axes[i-7].tick_params(labelsize=12)
                    axes[i-7].get_yaxis().set_label_coords(-0.2,0.5)
                    axes[i-7].xaxis.grid(True, which="minor")
                    axes[i-7].grid(which='major', axis='x', linestyle='--')

                for i in range(0,7):

                    axes[i+1].get_yaxis().set_label_coords(-0.01,0.5)
                    axes[i+1].right_ax.get_yaxis().set_label_coords(1.05, 0.5)
                    axes[i+1].xaxis.grid(True, which="minor")
                    axes[i+1].set_xlabel('Time', fontsize = 14)
```

```
        axes[i+1].tick_params(pad = 15)
        axes[i+1].grid(which='major', axis='x', linestyle='--')
        lines = axes[i+1].get_lines()

    fig.align_ylabels(axes[:])
    plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
    plt.savefig('_'  + file_name + '_' + task_name + '_' + str(x) + '.png', bbox_inches='tight')
    plt.savefig('_'  + file_name + '_' + task_name + '_' + str(x) + '.pdf', bbox_inches='tight')

    x += 1

    plt.show()

##########################################################################################################
##########################################################################################################
##########################################################################################################
                                                # EYE MOVEMENTS METRIC VISUALIZATION
##########################################################################################################
##########################################################################################################
##########################################################################################################
fc = synchronized_data.loc[synchronized_data['fixation'] == 2].groupby('AOI').size().sort_index(ascending=True)

fc = fc.reset_index()
fc.columns = ['AOI', 'Fixation Count']

fp = synchronized_data['AOI'].loc[synchronized_data['fixation'] == 2].value_counts(normalize = True).sort_index(ascendi
fp = fp.reset_index()
fp.columns = ['AOI', 'Fixation Count %']

tf = synchronized_data.loc[synchronized_data['fixation'] == 2].groupby('AOI')['Gaze event duration'].mean().sort_index(
tf = tf.reset_index()
tf.columns = ['AOI', 'Fixation Duration']

tp = (synchronized_data.loc[synchronized_data['fixation'] == 2].groupby('AOI')['Gaze event duration'].mean() / synchron
tp = tp.reset_index()
tp.columns = ['AOI', 'Fixation Duration %']

rt = synchronized_data.loc[(synchronized_data['fixation'] == 2)].groupby('AOI')['re_entry'].mean()
rt = rt.reset_index()
rt.columns = ['AOI', 'Re-entry Time']
```

```
a = pd.merge(fc, fp, on='AOI')
b = pd.merge(tf, tp, on='AOI')
a_b = pd.merge(a, b, on = 'AOI')
a_b = pd.merge(a_b, rt, on = 'AOI')
a_b

for i in AOI_list:
    if a_b['AOI'].str.contains(i).any() == False:
        s = pd.Series([i, 0, 0, 0, 0, np.nan], index=['AOI', 'Fixation Count', 'Fixation Count %', 'Fixation Duration',
        a_b = a_b.append(s, ignore_index=True)

a_b['Group'] = synchronized_data['Group'].iloc[0]
a_b['Number'] = synchronized_data['Number'].iloc[0]
a_b['Training'] = synchronized_data['Training'].iloc[0]
a_b['Event'] = synchronized_data['Event'].iloc[0]
a_b

fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(10, 14), sharex=True)

a_b.plot(kind = 'bar', x="AOI", y=['Fixation Count %'], color='deepskyblue', ax=axes[0], position = 1, width = 0.27)
a_b.plot(kind= 'bar', x="AOI", y=['Fixation Duration %'], color = 'red' , secondary_y=True, ax=axes[0], position = 0, w

axes[0].set_ylabel('Percentage', fontname="Arial", fontsize=14)
axes[0].set_xlabel('Areas of Interest (AOI)')

axes[0].set_ylim(0, 1.0)
axes[0].right_ax.set_ylim((0, 1.0))
axes[0].right_ax.set_ylabel('Percentage', fontname="Arial", fontsize=14, labelpad = 21.5)

a_b.plot(kind = 'bar', x="AOI", y=['Fixation Count'], color='deepskyblue', ax=axes[1], position = 1, width = 0.27)
a_b.plot(kind= 'bar', x="AOI", y=['Fixation Duration'], color = 'red' , secondary_y=True, ax=axes[1], position = 0, wid

axes[1].set_ylabel('Number', fontname="Arial", fontsize=14)

axes[1].set_ylim(0, 100)
axes[1].right_ax.set_ylim((0, 2500))
axes[1].right_ax.set_ylim((0, 2500))
axes[1].right_ax.set_ylabel('Millisecond', fontname="Arial", fontsize=14, labelpad = 12.5)


a_b.plot(kind = 'bar', x="AOI", y=['Re-entry Time'], color='deepskyblue', ax=axes[2], position = 1, width = 0.27)
```

```
axes[2].set_ylabel('Second', fontname="Arial", fontsize=14)
axes[2].set_xlabel('Areas of Interest (AOI)')

plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=.1, hspace=.1)

axes[2].set_ylim(0, 60)
axes[2].xaxis.label.set_size(13)

axes[0].yaxis.grid(True)
axes[1].yaxis.grid(True)
axes[2].yaxis.grid(True)

axes[0].xaxis.grid(True)
axes[1].xaxis.grid(True)
axes[2].xaxis.grid(True)

plt.savefig('_'  + file_name + '_' + task_name + '_' + 'Eye Movement.png')
plt.savefig('_'  + file_name + '_' + task_name + '_' + 'Eye Movement.pdf')
plt.show()

#########################################################################################################################
#########################################################################################################################
#########################################################################################################################
                                                    # EYE MOVEMENTS METRIC VISUALIZATION
#########################################################################################################################
#########################################################################################################################
#########################################################################################################################

export_date = time.strftime('%Y_%m_%d-%H_%M_%S')

synchronized_data.to_csv('_'  + file_name + '_' + task_name + '_' + export_date + '_.csv', encoding='utf-8')
#########################################################################################################################
#########################################################################################################################
#########################################################################################################################
############################################################################ PLOTTING EYE MOVEMENTS DATA AND FLIGHT DATA #######
#########################################################################################################################
#########################################################################################################################
#########################################################################################################################

master_dataframe = master_dataframe.append(synchronized_data, sort=False)
```

```
#            if task_name == 'Landing':
#                master_landing = master_landing.append(synchronized_data, sort=False)
#
#            elif task_name == 'Level_Turn':
#                master_level_turn = master_level_turn.append(synchronized_data, sort=False)

#            elif task_name == 'Level_Flight':
#                master_level_flight = master_level_flight.append(synchronized_data, sort=False)
#
#            else:
#                master_descent = master_descent.append(synchronized_data, sort=False)

        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################
                                                            # THE END OF FOR LOOP FOR EACH FLIGHT SCENARIO
        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################

# TIME OF EXPERTING DATA
if synchronized_data.empty:
    print('+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
    print('+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
    print('+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
    print('+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
    print('+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
    print('                          IT IS AN EMPTY DATAFRAME                                   ')
    print('+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
    print('+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
    print('+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
    print('+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')
    print('+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')


else:
    export_date = time.strftime('%Y_%m_%d-%H_%M_%S')

    # STORE ALL DATA IN THE NAME_DATAFRAME
    name_dataframe = 'X_MASTER_' + task_name + '_' + export_date + '.csv'
```

```
    master_dataframe = master_dataframe[['time' ,'timestamp', 'time of scenario', 'time bracket', 'Participant name', 'Skill', 'Gro
                                         'Vind, kias', 'VVI, fpm', 'elev, yoke1', 'ailrn, yoke1', 'ruddr, yoke1', '
                                         'X_transformed_airplane', 'Z_transformed_airplane', 'runway_vicinity', '__
                                         'signal_elev_detect', 'signal_elev_max_std', 'signal_elev_previous_flag',
                                         'signal_ailrn_detect', 'signal_ailrn_max_std', 'signal_ailrn_previous_flag
                                         'signal_ruddr_detect', 'signal_ruddr_max_std', 'signal_ruddr_previous_flag
                                         'area_elev', 'area_ailrn', 'area_ruddr', 'cal_time']]

    master_dataframe.to_csv(name_dataframe, encoding='utf-8')

#    if task_name == 'Landing':
#        master_landing = master_landing[['time' ,'timestamp', 'time of scenario', 'time bracket', 'Participant name', 'Skill', 'Gr
#                                         'Vind, kias', 'VVI, fpm', 'elev, yoke1', 'ailrn, yoke1', 'ruddr, yoke
#                                         'X_transformed_airplane', 'Z_transformed_airplane', 'runway_vicinity'
#                                         'signal_elev_max', 'signal_elev_min', 'signal_elev_detect', 'signal_e
#                                         'signal_ailrn_max', 'signal_ailrn_min', 'signal_ailrn_detect', 'signa
#
#        master_landing.to_csv(name_dataframe, encoding='utf-8')
#
#    elif task_name == 'Level_Turn':
#        master_level_turn = master_level_turn[['time' ,'timestamp', 'time of scenario', 'time bracket', 'Participant name', 'Skill
#                                         'Vind, kias', 'VVI, fpm', 'elev, yoke1', 'ailrn, yoke1', 'ruddr, yoke
#                                         'X_transformed_airplane', 'Z_transformed_airplane', 'runway_vicinity'
#                                         'signal_elev_max', 'signal_elev_min', 'signal_elev_detect', 'signal_e
#                                         'signal_ailrn_max', 'signal_ailrn_min', 'signal_ailrn_detect', 'signa
#
#        master_level_turn.to_csv(name_dataframe, encoding='utf-8')
#
#    elif task_name == 'Level_Flight':
#        master_level_flight = master_level_flight[['time' ,'timestamp', 'time of scenario', 'time bracket', 'Participant name', 'S
#                                         'Vind, kias', 'VVI, fpm', 'elev, yoke1', 'ailrn, yoke1', 'ruddr, yoke
#                                         'X_transformed_airplane', 'Z_transformed_airplane', 'runway_vicinity'
#                                         'signal_elev_max', 'signal_elev_min', 'signal_elev_detect', 'signal_e
#                                         'signal_ailrn_max', 'signal_ailrn_min', 'signal_ailrn_detect', 'signa
#
#        master_level_flight.to_csv(name_dataframe, encoding='utf-8')
#
#    else:
#        master_descent = master_descent[['time' ,'timestamp', 'time of scenario', 'time bracket', 'Participant name', 'Skill', 'Gr
#                                         'Vind, kias', 'VVI, fpm', 'elev, yoke1', 'ailrn, yoke1', 'ruddr, yoke
#                                         'X_transformed_airplane', 'Z_transformed_airplane', 'runway_vicinity'
```

```
#                                                    'signal_elev_max', 'signal_elev_min', 'signal_elev_detect', 'signal_e
#                                                    'signal_ailrn_max', 'signal_ailrn_min', 'signal_ailrn_detect', 'signa

#        master_descent.to_csv(name_dataframe, encoding='utf-8')

t1 = time.time()
total = t1-t0
print('Time for running the code (second): ', total)
# #%%
            ##########################################################################################################################
            ##########################################################################################################################
            ##########################################################################################################################
            ############################################################## AGGREGATED EYE MOVEMENTS DATA ##############################
            ##########################################################################################################################
            ##########################################################################################################################
            ##########################################################################################################################

#%%

names = ['X_MASTER_Descent_2019_06_24-00_39_38.csv', 'X_MASTER_Landing_2019_06_24-01_47_15.csv', 'X_MASTER_Level_Flight_2019_06_24-

counter_scenario = ['fixation_count','fixation_%', 'reentry_time', 'dwell_time', 'dwell_%']
for filename in names:

    synchronized_data = pd.read_csv(filename)
    synchronized_data = synchronized_data.loc[synchronized_data['Participant name'] != 'E01-E-B-LSDT'].loc[synchronized_data['Parti

    counter_scenario = ['fixation_count','fixation_%', 'reentry_time', 'dwell_time', 'dwell_%']

    number_row = 4
    number_col = 3

    for task in counter_scenario:

        fig, axes = plt.subplots(nrows= number_row, ncols= number_col, figsize=(10, 7), sharex=True, sharey=True)

        for i in range(number_row):
            for j in range(number_col):
                axes[i,j].set_xticklabels(['','Prior\nTreatment','Post\nTreatment',''], fontsize = 10)
                axes[i,j].xaxis.grid(True, linestyle=':')
                axes[i,j].yaxis.grid(True, linestyle=':')
```

```
        axes[i,j].spines['left'].set_linewidth(1)
        axes[i,j].spines['right'].set_linewidth(1)
        axes[i,j].spines['top'].set_linewidth(1)
        axes[i,j].spines['bottom'].set_linewidth(1)

        if task in {'fixation_count', 'reentry_time'} :
            axes[i,j].set_xlim(0.75, 2.25)
            axes[i,j].set_ylim(0, 40)
            axes[i,j].set_xticks(range(0,4))
            axes[i,j].set_yticks([0, 10 , 20, 30, 40])

        elif task in {'fixation_%', 'dwell_%'}:
            axes[i,j].set_xlim(0.75, 2.25)
            axes[i,j].set_ylim(0, 0.8)
            axes[i,j].set_xticks(range(0,4))
            axes[i,j].set_yticks([0, 0.2, 0.4, 0.6, 0.8])

        elif task == 'dwell_time':
            axes[i,j].set_xlim(0.75, 2.25)
            axes[i,j].set_ylim(0, 1600)
            axes[i,j].set_xticks(range(0,4))
            axes[i,j].set_yticks([0, 400, 800, 1200, 1600])

if task == 'fixation_%':
    fig.text(-0.01, 0.5, 'Fixation Count (%)', ha='center', va='center', rotation='vertical')
elif task == 'dwell_%':
    fig.text(-0.01, 0.5, 'Dwell Time (%)', ha='center', va='center', rotation='vertical')
elif task == 'fixation_count':
    fig.text(-0.01, 0.5, 'Fixation Count', ha='center', va='center', rotation='vertical')
elif task == 'dwell_time':
    fig.text(-0.01, 0.5, 'Dwell Time (millisecond)', ha='center', va='center', rotation='vertical')
elif task == 'reentry_time':
    fig.text(-0.01, 0.5, 'Re-entry Time (second)', ha='center', va='center', rotation='vertical')

plt.figtext(0.525, 1.025, 'Areas Of Interst (AOIs)', ha='center', va='center')

for aoi in AOI_list:
    if task == 'fixation_count':
        cb = synchronized_data[aoi].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group']=='Control'].loc[
        ca = synchronized_data[aoi].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group']=='Control'].loc[
        eb = synchronized_data[aoi].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group']=='Experimental']
```

```
    ea = synchronized_data[aoi].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group']=='Experimental']

elif task == 'fixation_%':
    cb = synchronized_data[aoi].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group']=='Control'].loc[
    ca = synchronized_data[aoi].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group']=='Control'].loc[
    eb = synchronized_data[aoi].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group']=='Experimental']
    ea = synchronized_data[aoi].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group']=='Experimental']

elif task == 'dwell_time':
    cb = synchronized_data['Gaze event duration'].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group'
    ca = synchronized_data['Gaze event duration'].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group'
    eb = synchronized_data['Gaze event duration'].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group'
    ea = synchronized_data['Gaze event duration'].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group'

elif task == 'dwell_%':
    cb = synchronized_data['Gaze event duration'].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group'
    ca = synchronized_data['Gaze event duration'].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group'
    eb = synchronized_data['Gaze event duration'].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group'
    ea = synchronized_data['Gaze event duration'].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group'

elif task == 'reentry_time':
    cb = synchronized_data['re_entry'].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group']=='Control
    ca = synchronized_data['re_entry'].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group']=='Control
    eb = synchronized_data['re_entry'].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group']=='Experim
    ea = synchronized_data['re_entry'].loc[synchronized_data['fixation'] == 2].loc[synchronized_data['Group']=='Experim

if aoi == 'AI':

    if task in {'fixation_%', 'dwell_%', 'reentry_time'}:
        q1 = axes[0,0].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'orangered', label = 'Con
        q2 = axes[0,0].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='gold', label = 'Exp. Grou

    elif task in {'fixation_count', 'dwell_time'}:
        q1 = axes[0,0].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'deepskyblue', label = 'C
        q2 = axes[0,0].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='limegreen', label = 'Exp.

    axes[0,0].title.set_text('AI')

if aoi == 'ALT':
    if task in {'fixation_%', 'dwell_%', 'reentry_time'}:
        q1 = axes[0,1].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'orangered', label = 'Con
```

```
            q2 = axes[0,1].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='gold', label = 'Exp. Grou

        elif task in {'fixation_count', 'dwell_time'}:
            q1 = axes[0,1].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'deepskyblue', label = 'C
            q2 = axes[0,1].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='limegreen', label = 'Exp.

        axes[0,1].title.set_text('ALT')

    if aoi == 'ASI':
        if task in {'fixation_%', 'dwell_%', 'reentry_time'}:
            q1 = axes[0,2].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'orangered', label = 'Con
            q2 = axes[0,2].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='gold', label = 'Exp. Grou

        elif task in {'fixation_count', 'dwell_time'}:
            q1 = axes[0,2].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'deepskyblue', label = 'C
            q2 = axes[0,2].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='limegreen', label = 'Exp.
        axes[0,2].title.set_text('ASI')

    if aoi == 'CONT':
        if task in {'fixation_%', 'dwell_%', 'reentry_time'}:
            q1 = axes[1,0].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'orangered', label = 'Con
            q2 = axes[1,0].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='gold', label = 'Exp. Grou

        elif task in {'fixation_count', 'dwell_time'}:
            q1 = axes[1,0].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'deepskyblue', label = 'C
            q2 = axes[1,0].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='limegreen', label = 'Exp.
        axes[1,0].title.set_text('CONT.')

    if aoi == 'GAG':
        if task in {'fixation_%', 'dwell_%', 'reentry_time'}:
            q1 = axes[1,1].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'orangered', label = 'Con
            q2 = axes[1,1].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='gold', label = 'Exp. Grou

        elif task in {'fixation_count', 'dwell_time'}:
            q1 = axes[1,1].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'deepskyblue', label = 'C
            q2 = axes[1,1].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='limegreen', label = 'Exp.

        axes[1,1].title.set_text('GAG')

    if aoi == 'HI':
        if task in {'fixation_%', 'dwell_%', 'reentry_time'}:
```

```
            q1 = axes[1,2].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'orangered', label = 'Con
            q2 = axes[1,2].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='gold', label = 'Exp. Grou

        elif task in {'fixation_count', 'dwell_time'}:
            q1 = axes[1,2].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'deepskyblue', label = 'C
            q2 = axes[1,2].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='limegreen', label = 'Exp.
        axes[1,2].title.set_text('HI')

    if aoi == 'OSW':
        if task in {'fixation_%', 'dwell_%', 'reentry_time'}:
            q1 = axes[2,0].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'orangered', label = 'Con
            q2 = axes[2,0].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='gold', label = 'Exp. Grou

        elif task in {'fixation_count', 'dwell_time'}:
            q1 = axes[2,0].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'deepskyblue', label = 'C
            q2 = axes[2,0].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='limegreen', label = 'Exp.
        axes[2,0].title.set_text('OSW')

    if aoi == 'RT':
        if task in {'fixation_%', 'dwell_%', 'reentry_time'}:
            q1 = axes[2,1].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'orangered', label = 'Con
            q2 = axes[2,1].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='gold', label = 'Exp. Grou

        elif task in {'fixation_count', 'dwell_time'}:
            q1 = axes[2,1].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'deepskyblue', label = 'C
            q2 = axes[2,1].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='limegreen', label = 'Exp.
        axes[2,1].title.set_text('RT')

    if aoi == 'TAC':
        if task in {'fixation_%', 'dwell_%', 'reentry_time'}:
            q1 = axes[2,2].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'orangered', label = 'Con
            q2 = axes[2,2].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='gold', label = 'Exp. Grou

        elif task in {'fixation_count', 'dwell_time'}:
            q1 = axes[2,2].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'deepskyblue', label = 'C
            q2 = axes[2,2].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='limegreen', label = 'Exp.
        axes[2,2].title.set_text('TAC')

    if aoi == 'TC':
        if task in {'fixation_%', 'dwell_%', 'reentry_time'}:
            q1 = axes[3,0].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'orangered', label = 'Con
```

```
            q2 = axes[3,0].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='gold', label = 'Exp. Grou

        elif task in {'fixation_count', 'dwell_time'}:
            q1 = axes[3,0].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'deepskyblue', label = 'C
            q2 = axes[3,0].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='limegreen', label = 'Exp.
        axes[3,0].title.set_text('TC')

    if aoi == 'VSI':
        if task in {'fixation_%', 'dwell_%', 'reentry_time'}:
            q1 = axes[3,1].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'orangered', label = 'Con
            q2 = axes[3,1].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='gold', label = 'Exp. Grou

        elif task in {'fixation_count', 'dwell_time'}:
            q1 = axes[3,1].plot([1,2], [cb,ca], linestyle='--', marker='o', linewidth=1.0, color= 'deepskyblue', label = 'C
            q2 = axes[3,1].plot([1,2], [eb,ea], linestyle='--', marker='^', linewidth=1.0, color='limegreen', label = 'Exp.
        axes[3,1].title.set_text('VSI')

axes[3,2].grid(False)
axes[3,2].set_xlim(0.75, 2.25)
plt.legend([axes[0,0].get_lines()[0], axes[0,0].get_lines()[1]], ['Control Grop' , 'Experimental Group'], loc='center', fra

plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.7)
plt.savefig('X_'  + filename[9:16] + '_' + str(task) + '.png', bbox_inches='tight')
plt.savefig('X_'  + filename[9:16] + '_' + str(task) + '.pdf', bbox_inches='tight')

plt.show()

    ##########################################################################################################################
    ##########################################################################################################################
    ##########################################################################################################################
    ############################################################################ FIXATION COUNT ###############################
    ##########################################################################################################################
    ##########################################################################################################################
    ##########################################################################################################################

                    ##########################################################################################################
                    ##########################################################################################################
                    ##########################################################################################################
                                                                                    # VISUAL ENTROPY PLOT
                    ##########################################################################################################
                    ##########################################################################################################
```

```
##########################################################################################################

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(15, 7), sharex=True)

dwell_df = pd.read_csv(filename)
entropy_df = pd.read_csv(filename)

entropy_df = entropy_df.drop_duplicates(subset=['Participant name', 'Training'], keep='first')

subject_name = dwell_df['Participant name'].unique().tolist()
group_name = dwell_df['Group'].unique().tolist()
entropy_df['TOI'] = np.nan

for name_ in subject_name:
    for group_ in group_name:

        entropy_df['TOI'].loc[(entropy_df['Participant name'] == name_) & (entropy_df['Group'] == group_)] = dwell_df['Gaze eve
        time.sleep(0)


entropy_df = entropy_df[['Participant name', 'Training', 'Group', 'entropy', 'TOI']]

entropy_df = entropy_df.reset_index(drop = True)
entropy_df
for index, row in entropy_df.iterrows():
    if row['Participant name'][0] == 'E':
        entropy_df = entropy_df.drop(entropy_df.index[index])

entropy_df = entropy_df.reset_index(drop = True)

entropy_df['Training'] = np.where(entropy_df['Participant name'].str[6:7]=='A', 'After', 'Before')


entropy_df['Group'] = entropy_df['Group'].str.replace('Control','Con.')
entropy_df['Group'] = entropy_df['Group'].str.replace('Experimental','Exp.')

con_group_range = entropy_df['Group'].loc[entropy_df['Group'] == 'Con.'].count()
exp_group_range = entropy_df['Group'].loc[entropy_df['Group'] == 'Exp.'].count()

entropy_df['no'] = np.nan
```

```
row_counter = 1
con_counter = 1

for index, row in entropy_df.iterrows():
    if index == entropy_df.index[-1]:
        break

    elif entropy_df['Group'].iloc[index] == 'Con.':
        if row_counter % 2 != 0:
            entropy_df['no'].iloc[index] = con_counter
            entropy_df['no'].iloc[index+1] = con_counter
            con_counter += 1



    elif entropy_df['Group'].iloc[index] == 'Exp.':
        if row_counter % 2 != 0:
            entropy_df['no'].iloc[index] = con_counter
            entropy_df['no'].iloc[index+1] = con_counter
            con_counter += 1

    row_counter += 1

entropy_df = entropy_df[['Participant name', 'no', 'Group', 'Training', 'entropy', 'TOI']]
entropy__expert_threshold = entropy_df['entropy'].iloc[0:2].mean()

#my_color=np.where((ordered_df['group']=='experimental') , 'orange', 'skyblue')
#my_size=np.where(ordered_df ['group']=='B', 70, 30)

axes.hlines(y=entropy_df['no'].loc[entropy_df['Training'] =='Before'],
           xmin=entropy_df['entropy'].loc[(entropy_df['Training'] =='Before')],
           xmax=entropy_df['entropy'].loc[(entropy_df['Training'] =='After')],
           color='grey', alpha=0.4, zorder=1, linewidth=2.0)

axes.axvline(entropy__expert_threshold, linestyle='dashed', linewidth=2.5, color = 'yellowgreen')
axes.text(x = entropy__expert_threshold + 0.05, y = (entropy_df['no'].max() +1) / 2, s='Expert Visual Entropy',  rotation=90, v

axes.axvline(-math.log2(110**-1), linestyle='dashed', linewidth=2.5, color = 'yellowgreen')
axes.text(x = 6.83, y = (entropy_df['no'].max() +1) / 2, s='Max. Value',  rotation=90, verticalalignment='center', color='grey'

axes.axvline(-math.log2(1**-1), linestyle='dashed', linewidth=2.5, color = 'yellowgreen')
```

```
axes.text(x = 0.05, y = (entropy_df['no'].max() +1) / 2 , s='Min. Value',  rotation=90, verticalalignment='center', color='grey

E1 = axes.scatter(entropy_df['entropy'].loc[(entropy_df['Training'] =='Before') & (entropy_df['Group'] =='Exp.')],
            entropy_df['no'].loc[(entropy_df['Training'] =='Before') & (entropy_df['Group'] =='Exp.')], color='pink', alpha=1.0

E2 = axes.scatter(entropy_df['entropy'].loc[(entropy_df['Training'] =='After') & (entropy_df['Group'] =='Exp.')],
            entropy_df['no'].loc[(entropy_df['Training'] =='After') & (entropy_df['Group'] =='Exp.')], color='orangered', alpha

E3 = axes.scatter(entropy_df['entropy'].loc[(entropy_df['Training'] =='Before') & (entropy_df['Group'] =='Con.')],
            entropy_df['no'].loc[(entropy_df['Training'] =='Before') & (entropy_df['Group'] =='Con.')], color='skyblue', alpha=

E4 = axes.scatter(entropy_df['entropy'].loc[(entropy_df['Training'] =='After') & (entropy_df['Group'] =='Con.')],
            entropy_df['no'].loc[(entropy_df['Training'] =='After') & (entropy_df['Group'] =='Con.')], color='blue', alpha=1, l

#plt.rc('grid', linestyle="-", color='black')
axes.set_yticklabels([])
# Add title and axis names
axes.grid()

axes.legend(loc="upper right")
axes.legend([E1, E2, E3, E4],['Exp. Group Prior Training', 'Exp. Group Post Training', 'Con. Group Prior Training', 'Con. Group
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)

plt.xlabel('Visual Entropy', fontweight='bold')
plt.ylabel('Participants', fontweight='bold')
plt.xlim(-0.05 , 7)
plt.yticks(np.arange(1, entropy_df['no'].max() +1 , 1.0))

task_name = 'Landing'

plt.savefig('X_' + filename[9:16] + 'visual_entropy.png', bbox_inches='tight')
plt.savefig('X_' + filename[9:16] + 'visual_entropy.pdf', bbox_inches='tight')

plt.grid(True)

plt.show()

    # https://seaborn.pydata.org/tutorial/axis_grids.html
    ##########################################################################################################################
    ##########################################################################################################################
    ##########################################################################################################################
```

```
                                                                    # VISUAL ENTROPY PLOT #
    ##########################################################################################################################################
    ##########################################################################################################################################
    ##########################################################################################################################################
    ##########################################################################################################################################
    ##########################################################################################################################################
    ##########################################################################################################################################
                                                                    #   LANIDING POINTS   #
    ##########################################################################################################################################
    ##########################################################################################################################################
    ##########################################################################################################################################

    # THE LANDING POINT FOR EXPERMENTAL AND CONTROL GROUP ==> BEFORE AND AFTER CONTROL
    # filename = 'X_MASTER_Landing_2019_06_18-17_37_48.csv'

if filename[9:16] == 'Landing':
    landing_points_all = pd.read_csv(filename)

    landing_points_all = landing_points_all.drop_duplicates(subset=['Participant name', 'Training'], keep='first')
    landing_points_all = landing_points_all[['Group', 'Training', 'x landing point', 'z landing point', 'x optimal landing poin

    fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(15, 7), sharex=True)


    landing_points_all

    #axes.scatter(x='x_transformed' , y= 'z_transformed', c='___on,runwy', data = cloud_points_runway)
    axes.vlines(15593.8841, ymin=16418.98266 , ymax=16449.29657, linestyle='-', linewidth=2.0, color='black', alpha=0.25)
    axes.vlines(16814.52156, ymin=16418.98266 , ymax=16449.29657, linestyle='-', linewidth=2.0, color = 'black', alpha=0.25)

    axes.vlines(landing_points_all['x optimal landing point'].iloc[0], ymin=16418.98266 , ymax=16449.29657, linestyle='-.', lin

    axes.hlines(16418.98266, xmin=15593.8841 , xmax=16814.52156 , linestyle='-', linewidth=2.0, color = 'black', alpha=0.25)
    axes.hlines(16449.29657, xmin=15593.8841 , xmax=16814.52156 , linestyle='-', linewidth=2.0, color = 'black', alpha=0.25)

    axes.hlines((16418.98266+16449.29657)/2, xmin=15593.8841 , xmax=16814.52156 , linestyle='-.', linewidth=2.0, color = 'black

    P0 = axes.scatter(landing_points_all['x optimal landing point'].iloc[0], landing_points_all['z optimal landing point'].iloc

    P1= axes.scatter(landing_points_all['x landing point'].loc[landing_points_all['Group'] == 'Experimental'].loc[landing_point
                landing_points_all['z landing point'].loc[landing_points_all['Group'] == 'Experimental'].loc[landing_points_al
```

```
                    s=80, facecolors='none', edgecolors='red', marker = 'o', color='red')

        P2= axes.scatter(landing_points_all['x landing point'].loc[landing_points_all['Group'] == 'Experimental'].loc[landing_point
                    landing_points_all['z landing point'].loc[landing_points_all['Group'] == 'Experimental'].loc[landing_points_al
                    s=80, edgecolors='tomato', marker = 'o', color='red', alpha=0.7)

        P3= axes.scatter(landing_points_all['x landing point'].loc[landing_points_all['Group'] == 'Control'].loc[landing_points_all
                    landing_points_all['z landing point'].loc[landing_points_all['Group'] == 'Control'].loc[landing_points_all['Tr
                    s=80, facecolors='none', marker = '^', color='blue')

        P4= axes.scatter(landing_points_all['x landing point'].loc[landing_points_all['Group'] == 'Control'].loc[landing_points_all
                    landing_points_all['z landing point'].loc[landing_points_all['Group'] == 'Control'].loc[landing_points_all['Tr
                    s=80, marker = '^', color = 'blue', alpha=0.8)

        axes.set_xlabel('X (meter)')
        axes.set_ylabel('Y (meter)')

        axes.legend(loc="upper right")
        axes.legend([P0, P1, P2, P3, P4],['Touchdown Point', 'Exp. Group Prior Training', 'Exp. Group Post Training', 'Con. Group P

        x = [15593.8841 , 16814.52156, 16814.52156, 15593.8841]
        y = [16418.98266 , 16418.98266 , 16449.29657 , 16449.29657]

        axes.fill_between(x, y, facecolor='black',alpha=0.05, lw=0)

        axes.set_xlim([15400, 17000])
        axes.set_ylim([16394.1396, 16474.1396])
        axes.grid()
        #plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)

        plt.savefig('X_landing_point.png', bbox_inches='tight')
        plt.savefig('X_landing_point.pdf', bbox_inches='tight')

        plt.show()

#%%
synchronized_data = pd.read_csv('X_MASTER_Landing_2019_06_24-01_47_15.csv')

plt.scatter(x=synchronized_data['X_transformed_airplane'].iloc[synchronized_data['X_transformed_airplane'].loc[synchronized_data['r
                                y=synchronized_data['Z_transformed_airplane'].iloc[synchronized_data['X_transformed_airplane'].lo
```

```
#%%
synchronized_data['Participant name'] == 'E01-E-B-LSDT'

#%%

synchronized_data['Y, m'].loc[synchronized_data['Participant name'] == 'E01-E-B-LSDT'].iloc[synchronized_data['X_transformed_airpla
#%%
synchronized_data['X_transformed_airplane'].loc[synchronized_data['Participant name'] == 'E01-E-B-LSDT'].iloc[synchronized_data['X_

synchronized_data['Z_transformed_airplane'].loc[synchronized_data['Participant name'] == 'E01-E-B-LSDT'].iloc[synchronized_data['X_

#%%
# 3D PLOT OF RUNWAY AND I WILL ADD FLIGHT PATH TO IT

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(30, 10))
ax = fig.add_subplot(111, projection='3d')

xs = cloud_points_runway['x_transformed'].loc[cloud_points_runway['___on,runwy']==0]
ys = cloud_points_runway['z_transformed'].loc[cloud_points_runway['___on,runwy']==0]
zs = cloud_points_runway['____Y,____m'].loc[cloud_points_runway['___on,runwy']==0]

ax.scatter(xs, ys, zs, s=5, alpha=0.5, edgecolors='skyblue', cmap='cubehelix')

xs = cloud_points_runway['x_transformed'].loc[cloud_points_runway['___on,runwy']==1]
ys = cloud_points_runway['z_transformed'].loc[cloud_points_runway['___on,runwy']==1]
zs = cloud_points_runway['____Y,____m'].loc[cloud_points_runway['___on,runwy']==1]
ax.scatter(xs, ys, zs, s=5, alpha=0.1, edgecolors='green', cmap='cubehelix')

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

ax.set_xlim([15400, 17000])
ax.set_ylim([16394.1396, 16474.1396])
ax.set_zlim([-21, 21])
ax.view_init(5, -90)

plt.show()
```

```
#%%
synchronized_data = pd.read_csv('X_MASTER_Landing_2019_06_24-01_47_15.csv')

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(30, 10))

ax = fig.add_subplot(111, projection='3d')
xs = cloud_points_runway['x_transformed'].loc[cloud_points_runway['___on,runwy']==0]
ys = cloud_points_runway['z_transformed'].loc[cloud_points_runway['___on,runwy']==0]
zs = cloud_points_runway['____Y,____m'].loc[cloud_points_runway['___on,runwy']==0]

ax.scatter(xs, ys, zs, s=5, alpha=0.5, edgecolors='skyblue', cmap='cubehelix')

xs = cloud_points_runway['x_transformed'].loc[cloud_points_runway['___on,runwy']==1]
ys = cloud_points_runway['z_transformed'].loc[cloud_points_runway['___on,runwy']==1]
zs = cloud_points_runway['____Y,____m'].loc[cloud_points_runway['___on,runwy']==1]
ax.scatter(xs, ys, zs, s=5, alpha=0.1, edgecolors='green', cmap='cubehelix')

for subjects in synchronized_data['Participant name'].loc[synchronized_data['Skill'] == 'Novice'].unique():
    xs = synchronized_data['X_transformed_airplane'].loc[synchronized_data['Participant name'] == subjects].loc[synchronized_data['
    ys = synchronized_data['Z_transformed_airplane'].loc[synchronized_data['Participant name'] == subjects].loc[synchronized_data['
    zs = synchronized_data['Y, m'].loc[synchronized_data['Participant name'] == subjects].loc[synchronized_data['Group']==group_].i
    ax.scatter(xs, ys, zs, s=5, alpha=0.1, edgecolors='red', cmap='cubehelix')

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

ax.set_xlim([15400, 17000])
ax.set_ylim([16394.1396, 16474.1396])
ax.set_zlim([-21, 21])
ax.view_init(5, -90)

plt.show()

#%%
synchronized_data = pd.read_csv('X_MASTER_Landing_2019_06_24-01_47_15.csv')

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
fig = plt.figure(figsize=(50, 10))

ax = fig.add_subplot(111, projection='3d')
xs = cloud_points_runway['x_transformed'].loc[cloud_points_runway['___on,runwy']==0]
ys = cloud_points_runway['z_transformed'].loc[cloud_points_runway['___on,runwy']==0]
zs = cloud_points_runway['____Y,____m'].loc[cloud_points_runway['___on,runwy']==0]

ax.scatter(xs, ys, zs, s=5, alpha=0.1, edgecolors='green', cmap='cubehelix')

xs = cloud_points_runway['x_transformed'].loc[cloud_points_runway['___on,runwy']==1]
ys = cloud_points_runway['z_transformed'].loc[cloud_points_runway['___on,runwy']==1]
zs = cloud_points_runway['____Y,____m'].loc[cloud_points_runway['___on,runwy']==1]
ax.scatter(xs, ys, zs, s=5, alpha=0.1, edgecolors='black', cmap='cubehelix')

for subjects in synchronized_data['Participant name'].loc[synchronized_data['Skill'] == 'Novice'].unique():
    for group_ in synchronized_data['Group'].loc[synchronized_data['Skill'] == 'Novice'].unique():
        for treatment in synchronized_data['Training'].loc[synchronized_data['Skill'] == 'Novice'].unique():
            xs = synchronized_data['X_transformed_airplane'].loc[synchronized_data['Participant name'] == subjects].loc[synchronize
            ys = synchronized_data['Z_transformed_airplane'].loc[synchronized_data['Participant name'] == subjects].loc[synchronize
            zs = synchronized_data['Y, m'].loc[synchronized_data['Participant name'] == subjects].loc[synchronized_data['Group']==g

            if group_ == 'Experimental' and treatment == 'Before Treatment':
                ax.scatter(xs, ys, zs, s=5, alpha=1.0, edgecolors='red', cmap='Dark2')
            elif group_ == 'Experimental' and treatment == 'After Treatment':
                ax.scatter(xs, ys, zs, s=5, alpha=0.25, edgecolors='red', cmap='Dark2')
            elif group_ == 'Control' and treatment == 'Before Treatment':
                ax.scatter(xs, ys, zs, s=5, alpha=1.0, edgecolors='purple', cmap='Dark2')
            elif group_ == 'Control' and treatment == 'After Treatment':
                ax.scatter(xs, ys, zs, s=5, alpha=0.25, edgecolors='purple', cmap='Dark2')

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

ax.set_xlim([15400, 17000])
ax.set_ylim([16394.1396, 16474.1396])
ax.set_zlim([-21, 21])
ax.view_init(5, -80)
plt.savefig('X_3D_landing_point.png', bbox_inches='tight')
plt.savefig('X_3D_landing_point.pdf', bbox_inches='tight')
plt.show()
```

```
#%%
    ##########################################################################################################################################
    ##########################################################################################################################################
    ##########################################################################################################################################
    ############################################################################ RESPONSE TIME PLOTS ############################################
    ##########################################################################################################################################
    ##########################################################################################################################################
    ##########################################################################################################################################

    style.available
    #style.use('ggplot')
    style.use('classic') #sets the size of the charts

   # RESPONSE TIME
    # https://stackoverflow.com/questions/43280854/python-matplotlib-normalising-multiple-plots-to-fit-the-same-arbitrary-axis-

    # https://stackoverflow.com/questions/35094454/how-would-one-use-kernel-density-estimation-as-a-1d-clustering-method-in-sci
    # TWO OBJECTIVES (1) TO PLOT ONE MINTUE OF DATA PER CHART (2) TO PLOT DEVIATIONS AS WELL
    # works fine for three plots, but now I want to add timeframe limit

    from matplotlib.dates import SecondLocator

    synchronized_data.to_csv('_'  + file_name + '_' + task_name + '_' + 'Synchronized_All.csv', encoding='utf-8')
    list_interval = ['00:00:00', '00:01:00', '00:02:00', '00:03:00', '00:04:00']

    x = 0
    while (x < len(list_interval)-1 and (synchronized_data[list_interval[x]:list_interval[x+1]]).empty == False):
        fig = plt.figure(figsize=(20, 10))

        ax = fig.add_subplot(211) # Create matplotlib axes

        ax.set_ymargin(0.025)
        ax.set_ylim([-1.0, 1.0])

        ax2 = ax.twinx() # Create another axes that shares the same x-axis as ax.
        ax2.spines['right'].set_position(('axes', 1.045))

        ax3 = ax.twinx() # Create another axes that shares the same x-axis as ax.
        ax3.spines['right'].set_position(('axes', 1.085))
```

```
synchronized_data['ailrn, yoke1'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (  synchronized_data['t
synchronized_data['max_signal_ailrn, yoke1'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (  synchroni
synchronized_data['min_signal_ailrn, yoke1'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (  synchroni

ax.axhline(y= 0.015, linestyle = '--', color='coral')
ax.axhline(y=-0.015, linestyle = '--', color='coral')

synchronized_data['cal rate turn'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (  synchronized_data['
synchronized_data['roll, deg'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (  synchronized_data['time
synchronized_data['deviation heading'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (  synchronized_da

ax.grid(which='major', axis='x', linestyle='--')
ax.right_ax.grid(which='major', axis='both', linestyle='--')


ax2.set_frame_on(True)
ax2.patch.set_visible(False)

    # Legends of plots
plt.legend([ax.get_lines()[0], ax.right_ax.get_lines()[0], ax2.get_lines()[0], ax3.get_lines()[0]], ['ailrn, yoke1' , '

    # Label of axis
ax.set_ylabel('Ailrn')
ax2.set_ylabel('Roll, deg.')
ax3.set_ylabel('Deviation Heading')
plt.ylabel('cal rate turn')

    # PART II
ax4 = fig.add_subplot(212) # Create matplotlib axes

ax4.set_ymargin(0.025)
ax4.set_ylim([-1.0, 1.0])

ax5 = ax4.twinx() # Create another axes that shares the same x-axis as ax.
ax5.spines['right'].set_position(('axes', 1.045))

ax6 = ax4.twinx() # Create another axes that shares the same x-axis as ax.
ax6.spines['right'].set_position(('axes', 1.085))

synchronized_data['elev, yoke1'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (  synchronized_data['ti
```

```
synchronized_data['max_signal_elev, yoke1'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (  synchroniz
synchronized_data['min_signal_elev, yoke1'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (  synchroniz

ax4.axhline(y= 0.015, linestyle = '--', color='coral')
ax4.axhline(y=-0.015, linestyle = '--', color='coral')

synchronized_data['deviation altitude'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (  synchronized_d
synchronized_data['pitch, deg'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (  synchronized_data['tim
synchronized_data['rpm n, engin'].loc[(synchronized_data['time bracket'] >= list_interval[x]) & (  synchronized_data['t

ax4.grid(which='major', axis='x', linestyle='--')
ax4.right_ax.grid(which='major', axis='both', linestyle='--')

ax5.set_frame_on(True)
ax5.patch.set_visible(False)

plt.legend([ax4.get_lines()[0], ax4.right_ax.get_lines()[0], ax5.get_lines()[0], ax6.get_lines()[0]], ['elev, yoke1' ,

ax4.set_ylabel('elev, yoke1')
ax5.set_ylabel('pitch, deg')
ax6.set_ylabel('rpm')
plt.ylabel('deviation altitude')

plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)

    #http://jonathansoma.com/lede/data-studio/matplotlib/adding-grid-lines-to-a-matplotlib-chart/
ax.set_axisbelow(True)

ax.minorticks_on()
ax4.minorticks_on()

ax.grid(axis='x', which='minor', linestyle=':', linewidth='0.5', color='black')
ax4.grid(axis='x', which='minor', linestyle=':', linewidth='0.5', color='black')

plt.savefig('_'  + file_name + '_' + task_name + '_' + str(x) + '_Response.png', bbox_inches='tight')
plt.savefig('_'  + file_name + '_' + task_name + '_' + str(x) + '_Response.pdf', bbox_inches='tight')

x += 1

plt.show()
```

```
        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################
        ############################################################################ RESPONSE TIME PLOTS ############################
        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################

    #%%
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# PRINTING ALL SIGNALS OF SCENARIO TIME IN ONE PLOT
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
synchronized_data.to_csv('_'  + file_name + '_' + task_name + '_' + 'Synchronized_All.csv', encoding='utf-8')

#%%
import pandas as pd
df = pd.read_csv('X_MASTER_Descent_2019_06_24-00_39_38.csv')
df = df.loc[df['fixation'] == 2]

a = df[['Group', 'Participant name', 'Training', 'Event', 'AOI', 'Gaze event duration', 'deviation altitude', 'cal rate turn']]
b = a.groupby(['Group', 'Training','Participant name', 'Event'])['deviation altitude'].agg(['mean', 'var', 'std']).unstack()
b.to_csv('zbbb.csv', encoding='utf-8')

#%%
b = b.sort_values([('Training')], ascending=False)
b = b.sort_values([('Group')], ascending=True)
#%%
b

#%%
        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################
        ############################################################################ FIXATION COUNT ############################
        ##########################################################################################################################
        ##########################################################################################################################
        ##########################################################################################################################

#my_colors = ['r','g','b','k']
a = synchronized_data.loc[synchronized_data['fixation'] == 2]
a = a[['Group', 'Training', 'AOI', 'Gaze event duration']]
```

```
b = a.groupby(['Group', 'Training','AOI']).count().unstack()
b = b.sort_values([('Training')], ascending=False)
b = b.sort_values([('Group')], ascending=True)
#b.plot(ax=axes[0,0], kind='bar', color=['r', 'b', 'g', 'y'], linewidth = 2)

b
#%%
axes[0,0].set_xticklabels(['Pre \nTreatment', 'Post\nTreatment', 'Pre\nTreatment', 'Post\nTreatment'], ha='center', rotation='horiz
axes[0,0].legend(AOI_list, loc='upper center', framealpha = 0.7, prop={'size': 10})
axes[0,0].set_xlabel('Control Group')
axes[0,0].set_ymargin(0.1)

plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)

#%%
fig, axes = plt.subplots(nrows=4, ncols=6, figsize=(10, 14), sharex=True)

#a_b.plot(kind = 'bar', x="AOI", y=['Fixation Count %'], color='deepskyblue', ax=axes[0], position = 1, width = 0.27)
#a_b.plot(kind= 'bar', x="AOI", y=['Fixation Duration %'], color = 'red' , secondary_y=True, ax=axes[0], position = 0, width = 0.27
#
#axes[0].set_ylabel('Percentage', fontname="Arial", fontsize=14)
#axes[0].set_xlabel('Areas of Interest (AOI)')
#axes[0].xaxis.grid(True)
#axes[0].set_ylim(0, 0.5)
#axes[0].right_ax.set_ylim((0, 0.5))
#axes[0].right_ax.set_ylabel('Percentage', fontname="Arial", fontsize=14, labelpad = 21.5)
#
#a_b.plot(kind = 'bar', x="AOI", y=['Fixation Count'], color='deepskyblue', ax=axes[1], position = 1, width = 0.27)
#a_b.plot(kind= 'bar', x="AOI", y=['Fixation Duration'], color = 'red' , secondary_y=True, ax=axes[1], position = 0, width = 0.27)
#
#axes[1].set_ylabel('Number', fontname="Arial", fontsize=14)
#axes[1].xaxis.grid(True)
#
#axes[1].set_ylim(0, 50)
#axes[1].right_ax.set_ylim((0, 2500))
#axes[1].right_ax.set_ylim((0, 2500))
#axes[1].right_ax.set_ylabel('Millisecond', fontname="Arial", fontsize=14, labelpad = 12.5)
#
#
#a_b.plot(kind = 'bar', x="AOI", y=['Re-entry Time'], color='deepskyblue', ax=axes[2], position = 1, width = 0.27)
#axes[2].set_ylabel('Second', fontname="Arial", fontsize=14)
```

```
#axes[2].set_xlabel('Areas of Interest (AOI)')
#
#plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=.1, hspace=.1)
#
#axes[2].xaxis.label.set_size(13)
#axes[2].xaxis.grid(True)
#
#axes[0].yaxis.grid(True)
#axes[1].yaxis.grid(True)
#axes[2].yaxis.grid(True)
plt.show()
#%%
```