biosignals_visualization_lab_1

November 27, 2020

1 Bio-signals Visualization Lab 1

1.1 Part 1: Movement signals

1.1.1 Load data

[2]:
```
import pandas as pd
from matplotlib import pyplot as plt
import scipy.io as spio
import numpy as np
import matplotlib.pyplot as plt

def get_duration_of_signal(data: pd.DataFrame) -> int:
    return data['time'].iloc[-1]
```

Table columns info structure

[26]:
```
csv_filed_infos = {
    "accel_x":      {"label": "ACCELEROMETER X (m/s²)",         "units": "m/
↪s²"},
    "accel_y":      {"label": "ACCELEROMETER Y (m/s²)",         "units": "m/
↪s²"},
    "accel_z":      {"label": "ACCELEROMETER Z (m/s²)",         "units": "m/
↪s²"},
    "grav_x":       {"label": "GRAVITY X (m/s²)",               "units": "m/
↪s²"},
    "grav_y":       {"label": "GRAVITY Y (m/s²)",               "units": "m/
↪s²"},
    "grav_z":       {"label": "GRAVITY Z (m/s²)",               "units": "m/
↪s²"},
    "lin_acc_x":    {"label": "LINEAR ACCELERATION X (m/s²)",   "units": "m/
↪s²"},
    "lin_acc_y":    {"label": "LINEAR ACCELERATION Y (m/s²)",   "units": "m/
↪s²"},
    "lin_acc_z":    {"label": "LINEAR ACCELERATION Z (m/s²)",   "units": "m/
↪s²"},
    "gyro_x":       {"label": "GYROSCOPE X (°/s)",              "units": "°/s"},
    "gyro_y":       {"label": "GYROSCOPE Y (°/s)",              "units": "°/s"},
```

```
    "gyro_z":       {"label": "GYROSCOPE Z (°/s)",              "units": "°"},
    "light":        {"label": "LIGHT (lux)",                   "units": "lux"},
    "magn_x":       {"label": "MAGNETIC FIELD X ( T)",         "units": " T"},
    "magn_y":       {"label": "MAGNETIC FIELD Y ( T)",         "units": " T"},
    "magn_z":       {"label": "MAGNETIC FIELD Z ( T)",         "units": " T"},
    "orien_z":      {"label": "ORIENTATION Z (azimuth °)",     "units":␣
 ↪"azimuth °"},
    "orien_x":      {"label": "ORIENTATION X (pitch °)",       "units": "pitch␣
 ↪°"},
    "orien_y":      {"label": "ORIENTATION Y (roll °)",        "units": "roll␣
 ↪°"},
    "prox":         {"label": "PROXIMITY (m)",                 "units": "m"},
    "sound":        {"label": "SOUND LEVEL (dB)",              "units": "dB"},
    "latitude":     {"label": "LOCATION Latitude",             "units":␣
 ↪"latitude"},
    "longitude":    {"label": "LOCATION Longitude",            "units":␣
 ↪"longitude"},
    "altitude":     {"label": "LOCATION Altitude ( m)",        "units": "m"},
    "altit_google": {"label": "LOCATION Altitude-google ( m)", "units": "m"},
    "speed":        {"label": "LOCATION Speed ( m/s)",         "units": "m/s"},
    "accuracy":     {"label": "LOCATION Accuracy ( m)",        "units": "m"},
    "orient":       {"label": "LOCATION ORIENTATION (°)",      "units": "°"},
    "satellites":   {"label": "Satellites in range",           "units":␣
 ↪"sat_num"},
    "time":         {"label": "Time since start in ms",        "units": "ms"},
    "time_stamp":   {"label": "YYYY-MO-DD HH-MI-S_SSS",        "units":␣
 ↪"YYYY-MO-DD HH-MI-S_SSS"}
}
```

[27]:
```
data_dir = "../data/adro_sensor_rec/"
run_and_stay = pd.read_csv(data_dir + "run_and_stay.csv")
stay_one_minute = pd.read_csv(data_dir + "stay_one_minute.csv")
walk_and_stay = pd.read_csv(data_dir + "walk_and_stay.csv")

print("stay_one_minute duration:", get_duration_of_signal(stay_one_minute),␣
 ↪csv_filed_infos['time']['units'])
print("walk_and_stay duration:".ljust(25),␣
 ↪get_duration_of_signal(walk_and_stay), csv_filed_infos['time']['units'])
print("run_and_stay duration:".ljust(25), get_duration_of_signal(run_and_stay),␣
 ↪csv_filed_infos['time']['units'])


run_and_stay.head(5)
```

```
stay_one_minute duration: 113675 ms
walk_and_stay duration:   47362 ms
run_and_stay duration:    47698 ms
```

[27]:
```
   accel_x  accel_y  accel_z  grav_x  grav_y  grav_z  lin_acc_x  lin_acc_y  \
0  -4.0299    5.361   7.2808 -3.8657  5.8201  6.8814    -0.1754    -0.4423
1  -4.1783    4.746   7.9005 -3.9739  5.8123  6.8262    -0.1822    -1.0385
2  -4.1783    4.746   7.9005 -3.9739  5.8123  6.8262    -0.1822    -1.0385
3  -4.1783    4.746   7.9005 -3.9739  5.8123  6.8262    -0.1822    -1.0385
4  -4.4367    5.026   6.7783 -4.0347  5.7421  6.8498    -0.4138    -0.6993

   lin_acc_z  gyro_x  …  latitude  longitude  altitude  altit_google  speed  \
0     0.4192  -13.70  …       NaN        NaN       NaN           NaN    NaN
1     1.1036  -30.46  …       NaN        NaN       NaN           NaN    NaN
2     1.1036  -30.46  …       NaN        NaN       NaN           NaN    NaN
3     1.1036  -30.46  …       NaN        NaN       NaN           NaN    NaN
4    -0.0514  -46.14  …       NaN        NaN       NaN           NaN    NaN

   accuracy  orient  satellites  time               time_stamp
0       NaN     NaN       0 / 0     8  2020-10-20 19:54:44:576
1       NaN     NaN       0 / 0    13  2020-10-20 19:54:44:581
2       NaN     NaN       0 / 0    18  2020-10-20 19:54:44:586
3       NaN     NaN       0 / 0    22  2020-10-20 19:54:44:590
4       NaN     NaN       0 / 0    28  2020-10-20 19:54:44:596

[5 rows x 31 columns]
```

[28]:
```
run_and_stay.describe()
```

[28]:
```
           accel_x      accel_y      accel_z       grav_x       grav_y  \
count  9539.000000  9539.000000  9539.000000  9539.000000  9539.000000
mean     -0.487243    -7.932248     0.210890    -0.498781    -7.502258
std       5.142110     5.019506     4.612677     4.734997     2.953805
min     -29.688200   -78.335100   -62.426900    -7.969100    -9.741300
25%      -5.494300    -8.905000    -0.967300    -5.504100    -8.792600
50%       0.628900    -8.440800    -0.766300     0.670600    -8.085500
75%       4.321000    -8.029200    -0.658600     4.278200    -8.033700
max      32.412800    19.033600    78.304600     6.546000     5.820100

            grav_z    lin_acc_x    lin_acc_y    lin_acc_z       gyro_x  …  \
count  9539.000000  9539.000000  9539.000000  9539.000000  9539.000000  …
mean      0.218080    -0.022214    -0.326688    -0.013730    -0.606059  …
std       2.906741     2.285907     3.226497     3.405825    64.149116  …
min      -5.015400   -28.370100   -35.951500   -46.183400  -719.160000  …
25%      -0.995800    -0.050700    -0.045200    -0.046100    -0.710000  …
50%      -0.780900    -0.003500    -0.006800     0.005200     0.020000  …
75%      -0.662200     0.045300     0.018900     0.066100     0.790000  …
max       9.800700    34.906000    26.164600    43.665100   551.350000  …

             prox        sound  latitude  longitude  altitude  altit_google  \
count  9539.000000  9539.000000       0.0        0.0       0.0           0.0
```

```
mean      0.502673    19.991133       NaN        NaN       NaN           NaN
std       1.503637    14.585172       NaN        NaN       NaN           NaN
min       0.000000     5.947000       NaN        NaN       NaN           NaN
25%       0.000000     9.000000       NaN        NaN       NaN           NaN
50%       0.000000    13.971000       NaN        NaN       NaN           NaN
75%       0.000000    24.072000       NaN        NaN       NaN           NaN
max       5.000000    59.560000       NaN        NaN       NaN           NaN

       speed  accuracy  orient          time
count    0.0       0.0     0.0   9539.000000
mean     NaN       NaN     NaN  23852.749554
std      NaN       NaN     NaN  13769.081407
min      NaN       NaN     NaN      8.000000
25%      NaN       NaN     NaN  11930.000000
50%      NaN       NaN     NaN  23852.000000
75%      NaN       NaN     NaN  35775.000000
max      NaN       NaN     NaN  47698.000000

[8 rows x 29 columns]
```

1.1.2 Clean the data

Delete unused data, and empty data columns from the dataset.

[29]:
```
run_and_stay.isnull().sum()
```

[29]:
```
accel_x           0
accel_y           0
accel_z           0
grav_x            0
grav_y            0
grav_z            0
lin_acc_x         0
lin_acc_y         0
lin_acc_z         0
gyro_x            0
gyro_y            0
gyro_z            0
light             0
magn_x            0
magn_y            0
magn_z            0
orien_z           0
orien_x           0
orien_y           0
prox              0
sound             0
```

```
latitude        9539
longitude       9539
altitude        9539
altit_google    9539
speed           9539
accuracy        9539
orient          9539
satellites         0
time               0
time_stamp         0
dtype: int64
```

Drop localization columns, as the GPS module was probably not enabled.

[30]:
```
columns_to_drop = ["latitude", "longitude", "altitude", "altit_google",␣
↪"speed", "accuracy", "orient"]
run_and_stay.drop(columns=columns_to_drop)
stay_one_minute.drop(columns=columns_to_drop)
walk_and_stay.drop(columns=columns_to_drop)

run_and_stay.columns
```

[30]:
```
Index(['accel_x', 'accel_y', 'accel_z', 'grav_x', 'grav_y', 'grav_z',
       'lin_acc_x', 'lin_acc_y', 'lin_acc_z', 'gyro_x', 'gyro_y', 'gyro_z',
       'light', 'magn_x', 'magn_y', 'magn_z', 'orien_z', 'orien_x', 'orien_y',
       'prox', 'sound', 'latitude', 'longitude', 'altitude', 'altit_google',
       'speed', 'accuracy', 'orient', 'satellites', 'time', 'time_stamp'],
      dtype='object')
```

Select needed columns.

[31]:
```
run_and_stay = run_and_stay[[
                 'accel_x','accel_y', 'accel_z',
                 'gyro_x', 'gyro_y', 'gyro_z',
                 'time']]
stay_one_minute = stay_one_minute[[
                 'accel_x','accel_y', 'accel_z',
                 'gyro_x', 'gyro_y', 'gyro_z',
                 'time']]
walk_and_stay = walk_and_stay[[
                 'accel_x','accel_y', 'accel_z',
                 'gyro_x', 'gyro_y', 'gyro_z',
                 'time']]

walk_and_stay.head(5)
```

[31]:

	accel_x	accel_y	accel_z	gyro_x	gyro_y	gyro_z	time
0	1.5693	0.8936	7.7019	-5.44	105.14	10.46	16

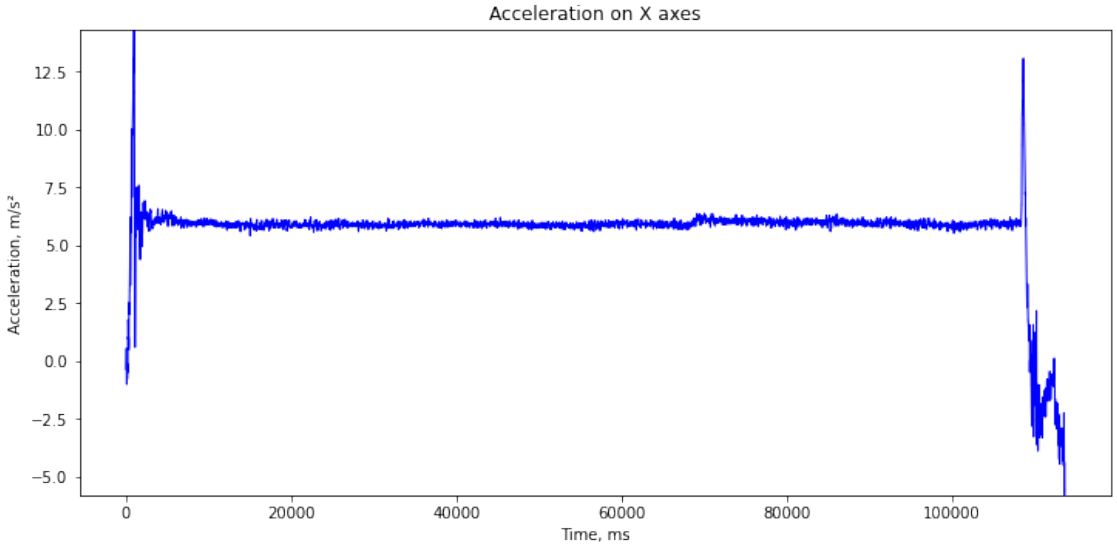
```
1   1.5693   0.8936   7.7019   -5.44   105.14   10.46   27
2   1.5693   0.8936   7.7019   -5.44   105.14   10.46   28
3   1.5693   0.8936   7.7019   -5.44   105.14   10.46   28
4   1.5693   0.8936   7.7019   -5.44   105.14   10.46   28
```

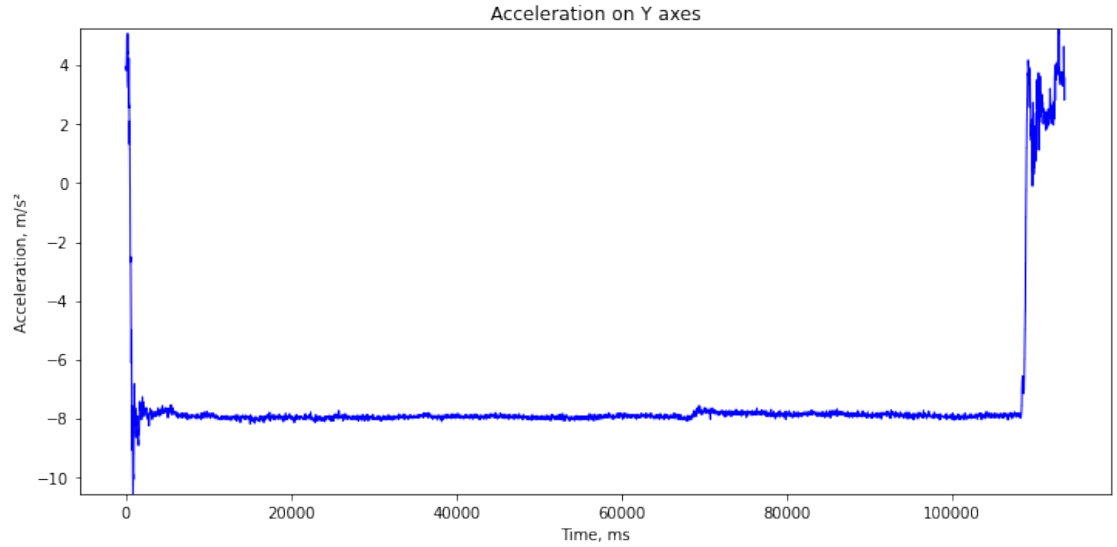
1.1.3 Accelerometer data plotting

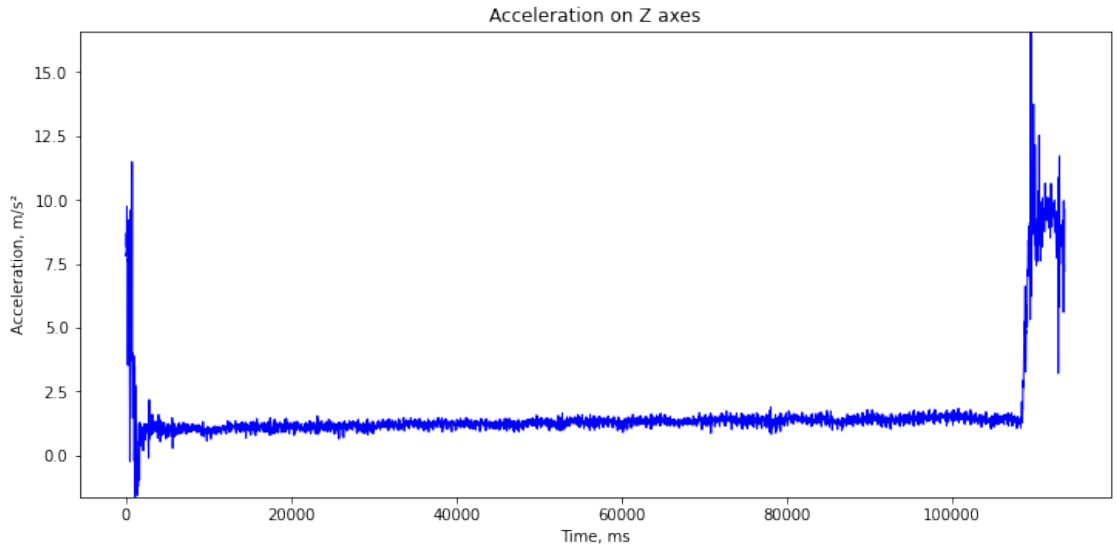
[33]:
```python
def plot_accel(dataset: pd.DataFrame, ax_local, n_prefix: str):
    time = dataset['time'].to_numpy()
    dataset = dataset[[f'accel_{n_prefix}']].to_numpy()
    ax_local.plot(time, dataset, 'b-', linewidth=1)

    ax_local.set_title(f'Acceleration on {n_prefix.upper()} axes')
    ax_local.set_xlabel(f'Time, {csv_filed_infos["time"]["units"]}')
    ax_local.set_ylabel(f'Acceleration, {csv_filed_infos[f"accel_{n_prefix.
 ↪lower()}"]["units"]}')   # relative to plt.rcParams['font.size']
    ax_local.set(ylim=(dataset.min(), dataset.max()))
```

Stay one minute data

[34]:
```python
_fig, ax1 = plt.subplots(nrows=3, ncols=1, constrained_layout=True,␣
 ↪figsize=(10,15))

plot_accel(stay_one_minute, ax1[0], 'x')
plot_accel(stay_one_minute, ax1[1], 'y')
plot_accel(stay_one_minute, ax1[2], 'z')

plt.show()
```

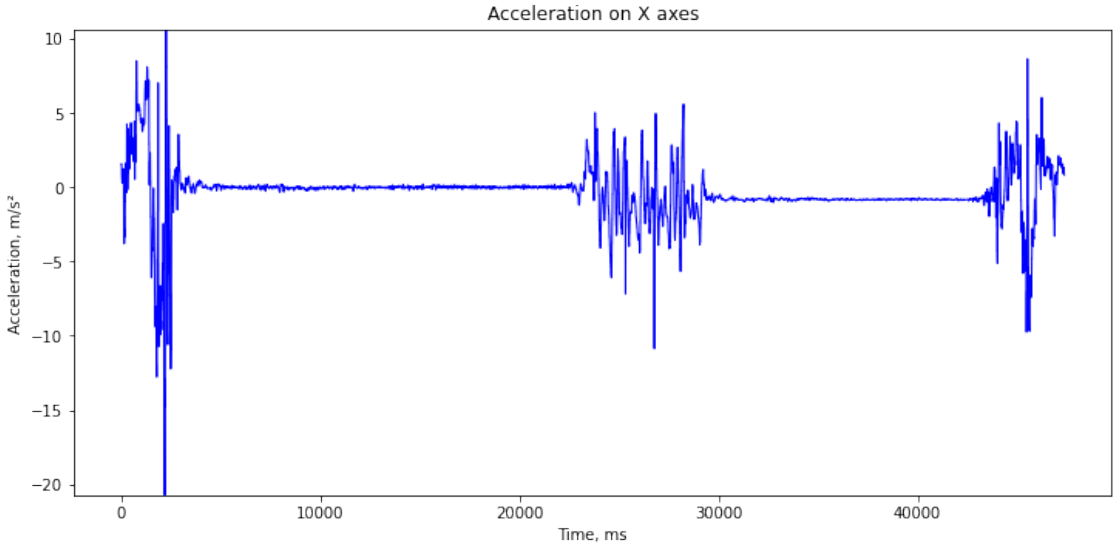

Acceleration on X axes
Acceleration, m/s²
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0
20000
40000
60000
80000
100000
Time, ms

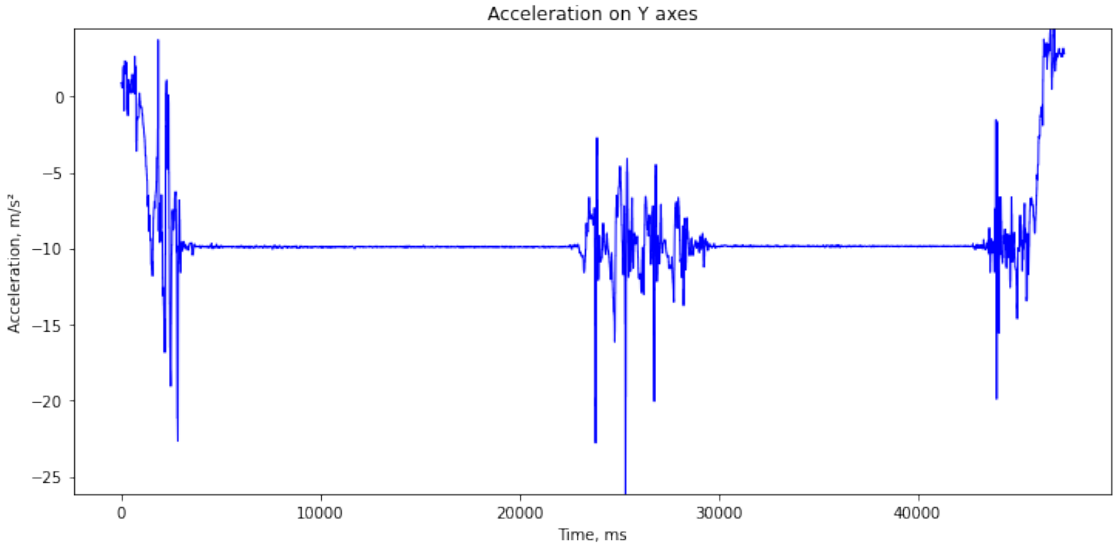

Acceleration on Y axes
Acceleration, m/s²
4
2
0
−2
−4
−6
−8
−10
0
20000
40000
60000
80000
100000
Time, ms

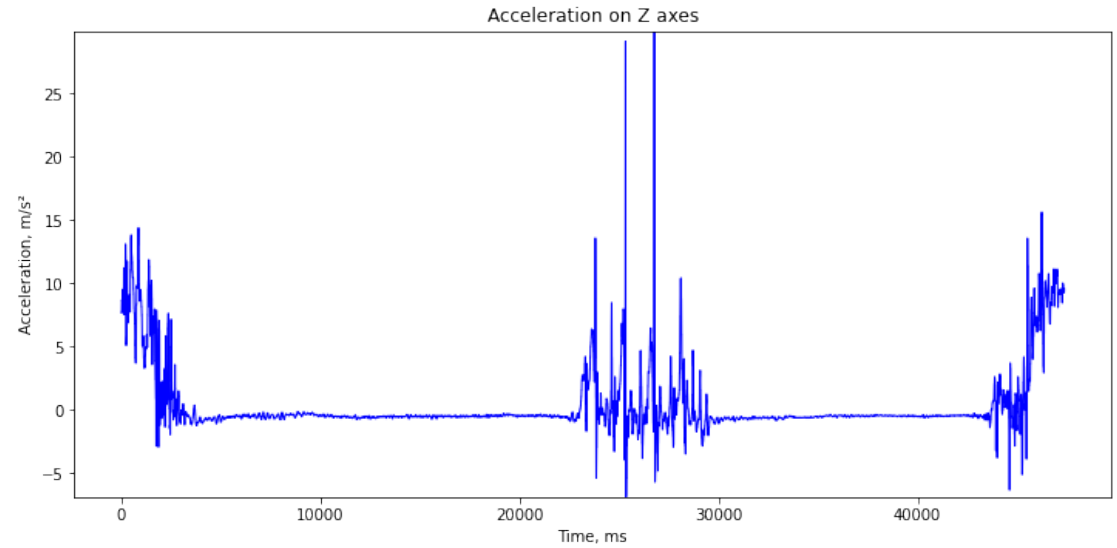

Acceleration on Z axes
Acceleration, m/s²
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
20000
40000
60000
80000
100000
Time, ms

Walk and Stay data

[35]:
```
_fig, ax2 = plt.subplots(nrows=3, ncols=1, constrained_layout=True,␣
↪figsize=(10,15))

plot_accel(walk_and_stay, ax2[0], 'x')
plot_accel(walk_and_stay, ax2[1], 'y')
plot_accel(walk_and_stay, ax2[2], 'z')

plt.show()
```

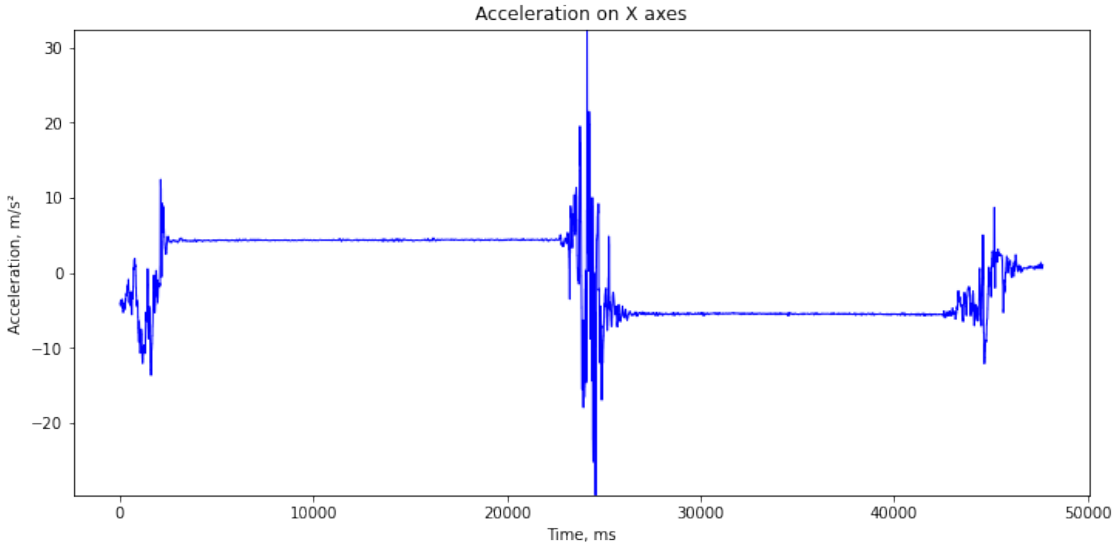

Acceleration on X axes
Acceleration, m/s²
Time, ms

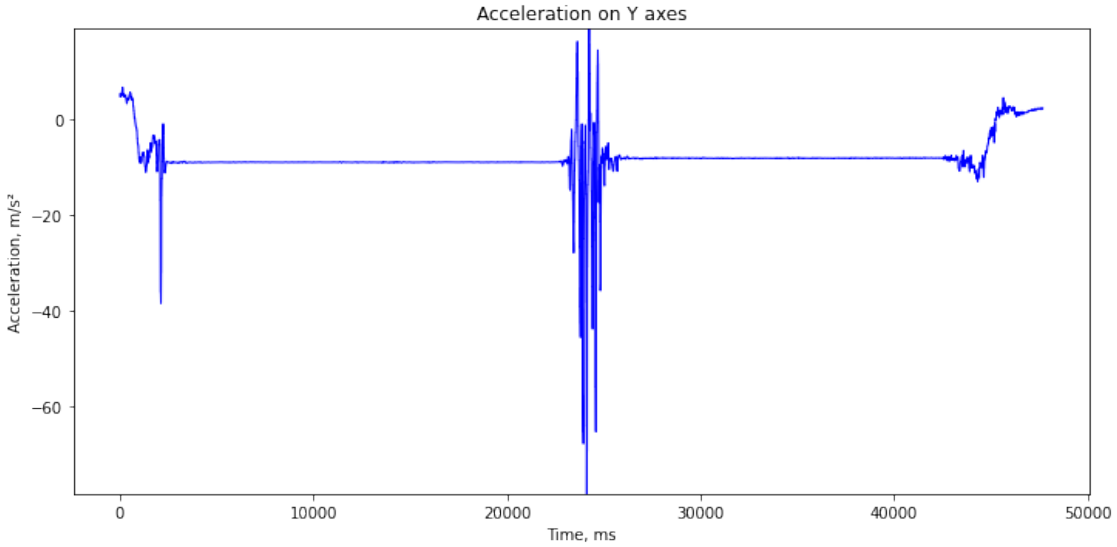

Acceleration on Y axes
Acceleration, m/s²
Time, ms

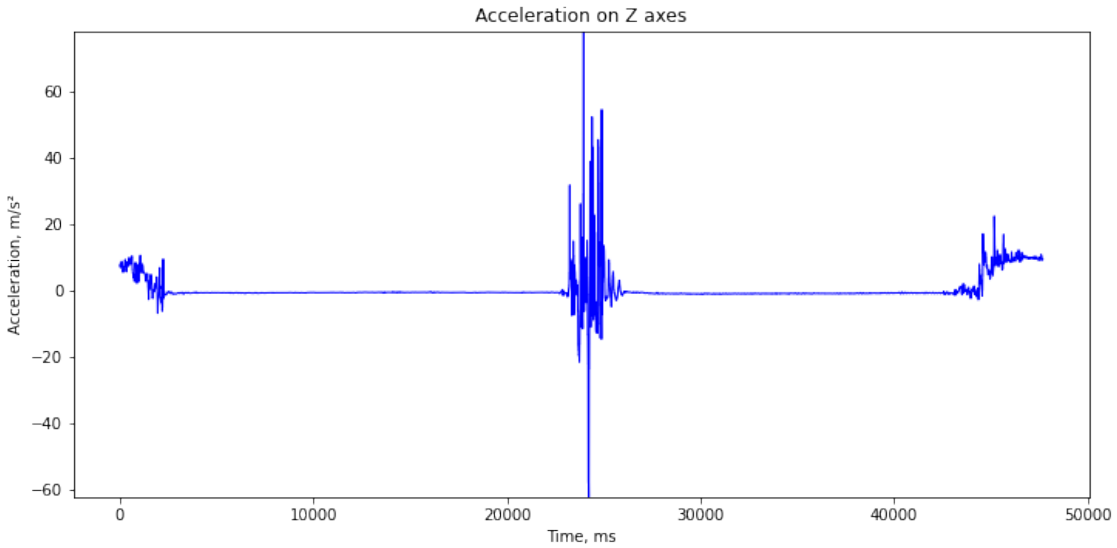

Acceleration on Z axes
Acceleration, m/s²
Time, ms

Run and Stay data

[36]:
```
_fig, ax3 = plt.subplots(nrows=3, ncols=1, constrained_layout=True,␣
↪figsize=(10,15))

plot_accel(run_and_stay, ax3[0], 'x')
plot_accel(run_and_stay, ax3[1], 'y')
plot_accel(run_and_stay, ax3[2], 'z')

plt.show()
```


Acceleration on X axes
Acceleration, m/s²
30
20
10
0
−10
−20
0
10000
20000
30000
40000
50000
Time, ms


Acceleration on Y axes
Acceleration, m/s²
0
−20
−40
−60
0
10000
20000
30000
40000
50000
Time, ms

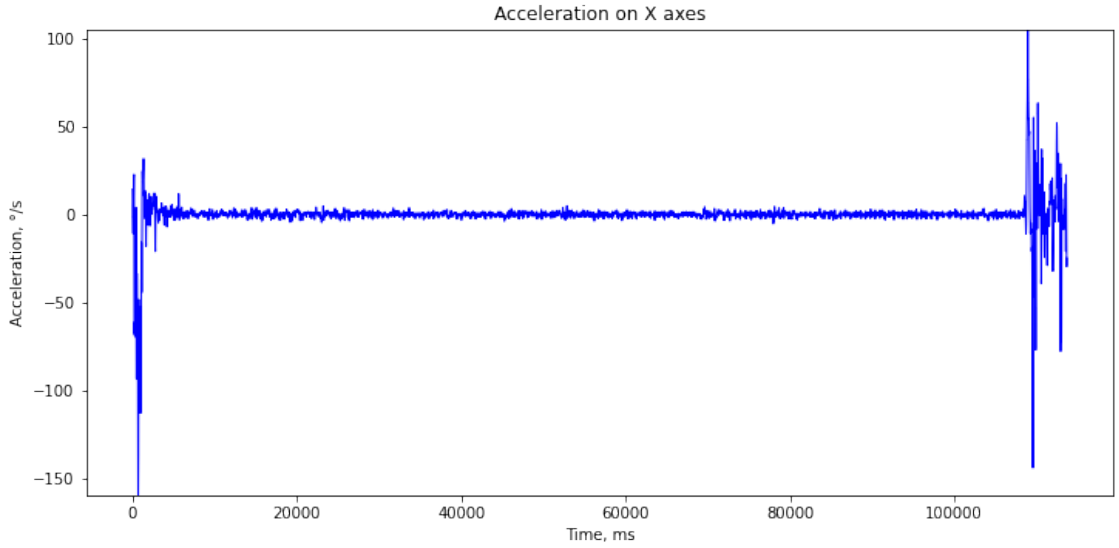

Acceleration on Z axes
Acceleration, m/s²
60
40
20
0
−20
−40
−60
0
10000
20000
30000
40000
50000
Time, ms

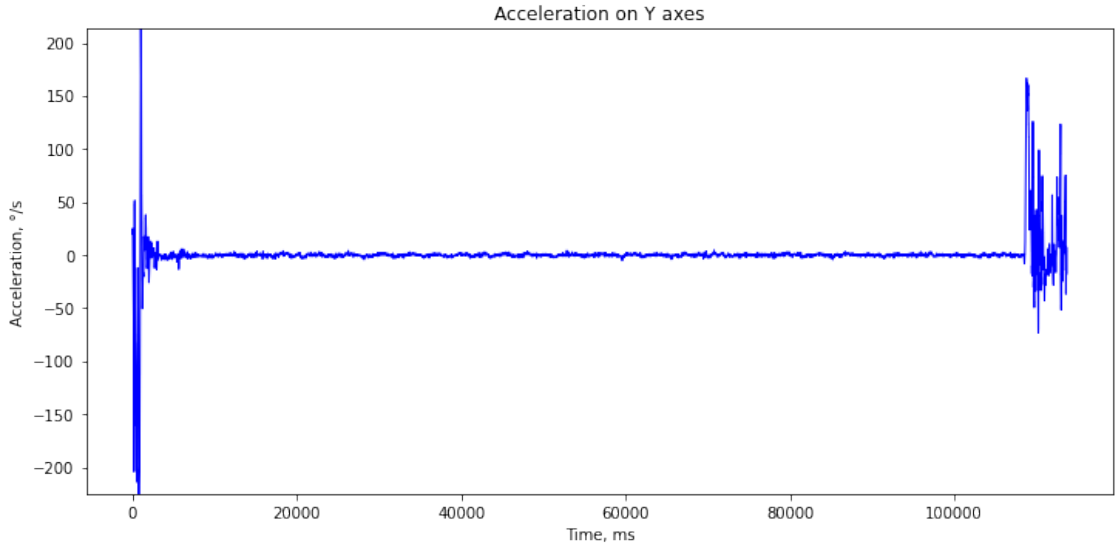
1.1.4 Gyroscope data plotting

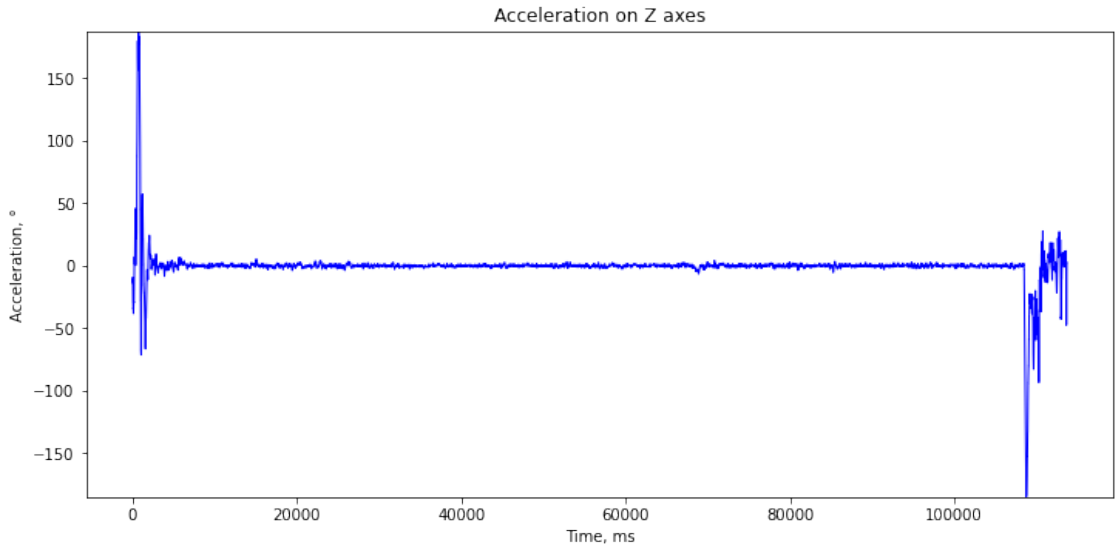
[37]:
```
def plot_gyro(dataset: pd.DataFrame, ax_local, n_prefix: str):
    time = dataset['time'].to_numpy()
    dataset = dataset[[f'gyro_{n_prefix}']].to_numpy()
    ax_local.plot(time, dataset, 'b-', linewidth=1)

    ax_local.set_title(f'Acceleration on {n_prefix.upper()} axes')
    ax_local.set_xlabel(f'Time, {csv_filed_infos["time"]["units"]}')
    ax_local.set_ylabel(f'Acceleration, {csv_filed_infos[f"gyro_{n_prefix.
 ↪lower()}"]["units"]}')   # relative to plt.rcParams['font.size']
    ax_local.set(ylim=(dataset.min(), dataset.max()))
```

Stay one minute data

[38]:
```
_fig, ax1 = plt.subplots(nrows=3, ncols=1, constrained_layout=True,␣
 ↪figsize=(10,15))

plot_gyro(stay_one_minute, ax1[0], 'x')
plot_gyro(stay_one_minute, ax1[1], 'y')
plot_gyro(stay_one_minute, ax1[2], 'z')

plt.show()
```

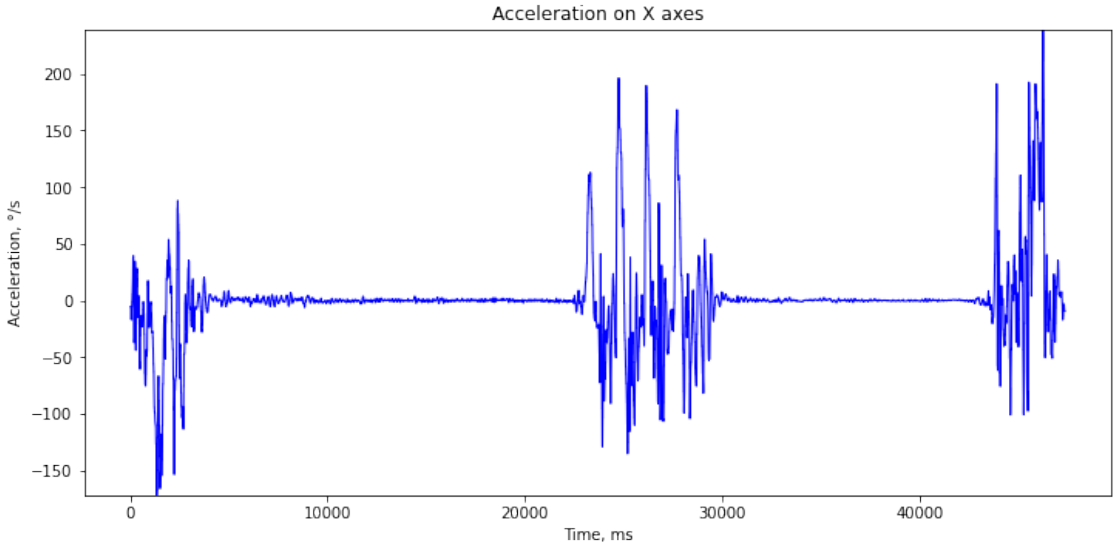

Acceleration on X axes
Acceleration, °/s
Time, ms

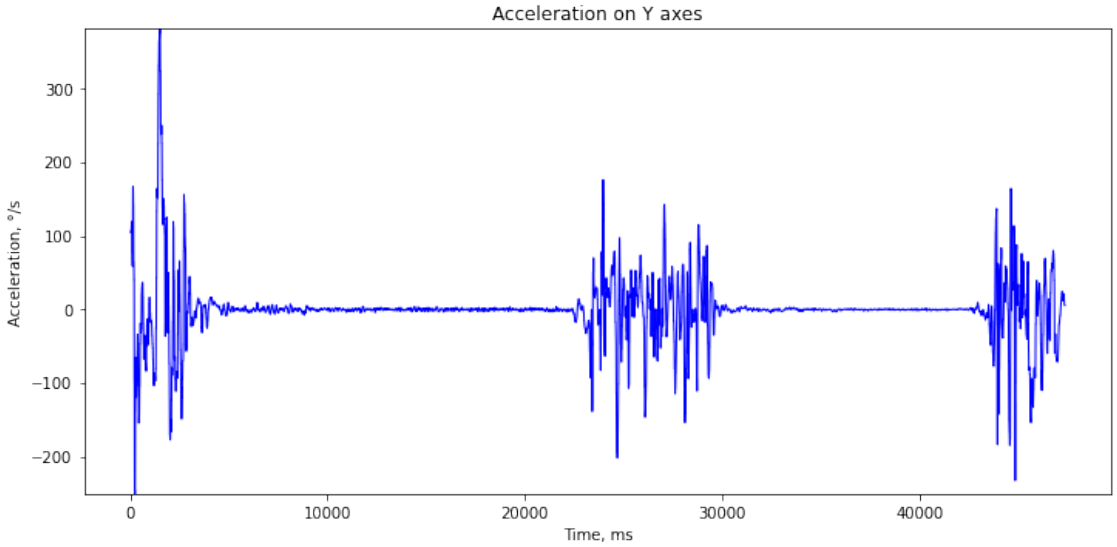

Acceleration on Y axes
Acceleration, °/s
Time, ms

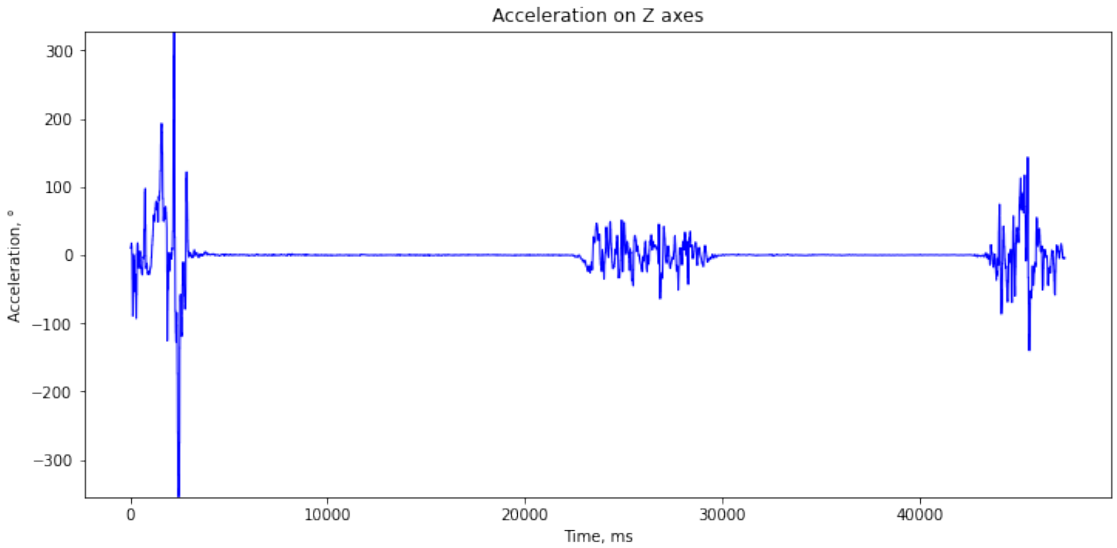

Acceleration on Z axes
Acceleration, °
Time, ms

Walk and Stay data

[39]:
```
_fig, ax2 = plt.subplots(nrows=3, ncols=1, constrained_layout=True,␣
↪figsize=(10,15))

plot_gyro(walk_and_stay, ax2[0], 'x')
plot_gyro(walk_and_stay, ax2[1], 'y')
plot_gyro(walk_and_stay, ax2[2], 'z')

plt.show()
```

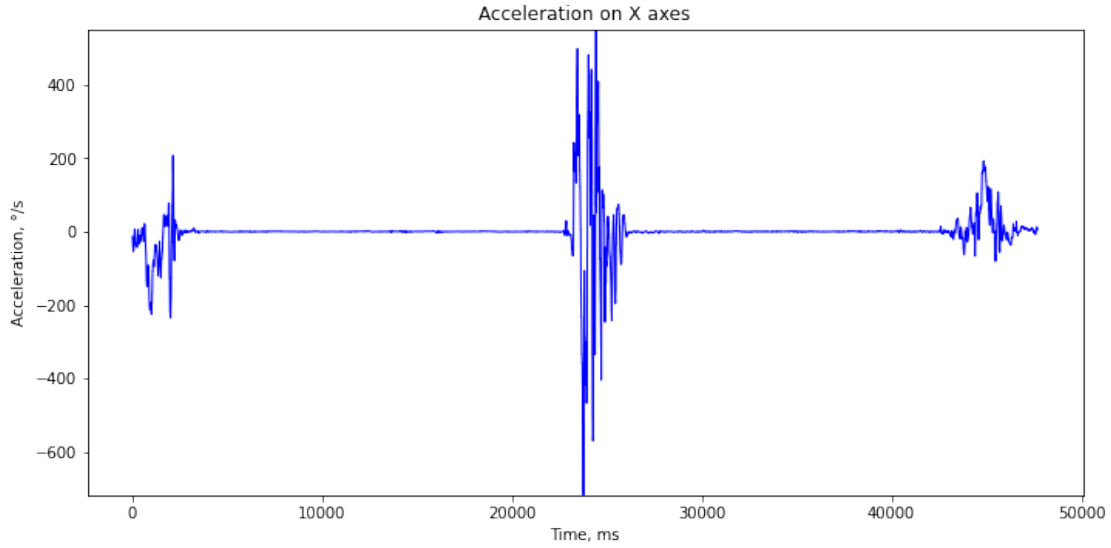

Acceleration on X axes
Acceleration, °/s
200
150
100
50
0
−50
−100
−150
0
10000
20000
30000
40000
Time, ms

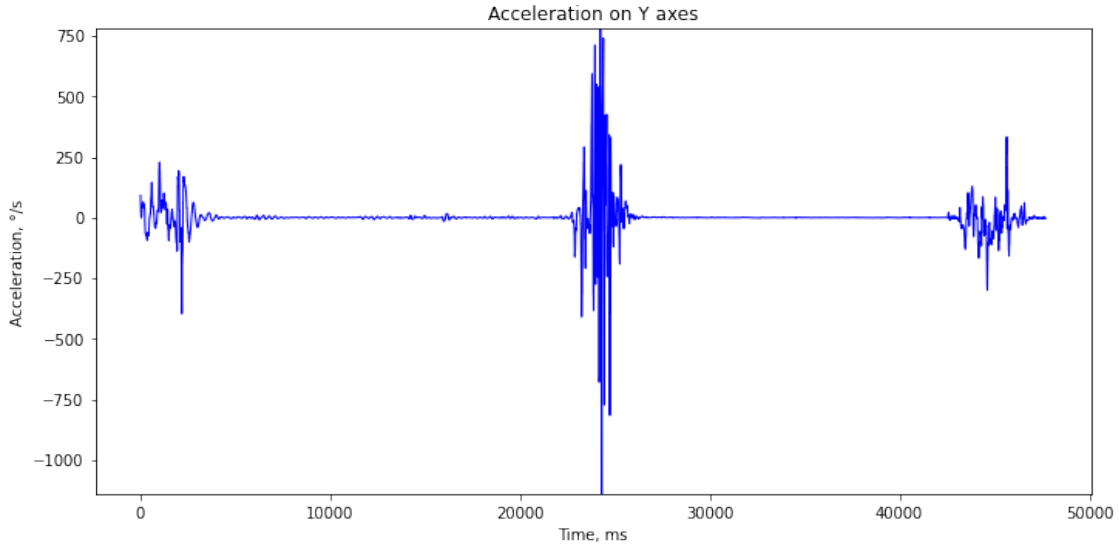

Acceleration on Y axes
Acceleration, °/s
300
200
100
0
−100
−200
0
10000
20000
30000
40000
Time, ms

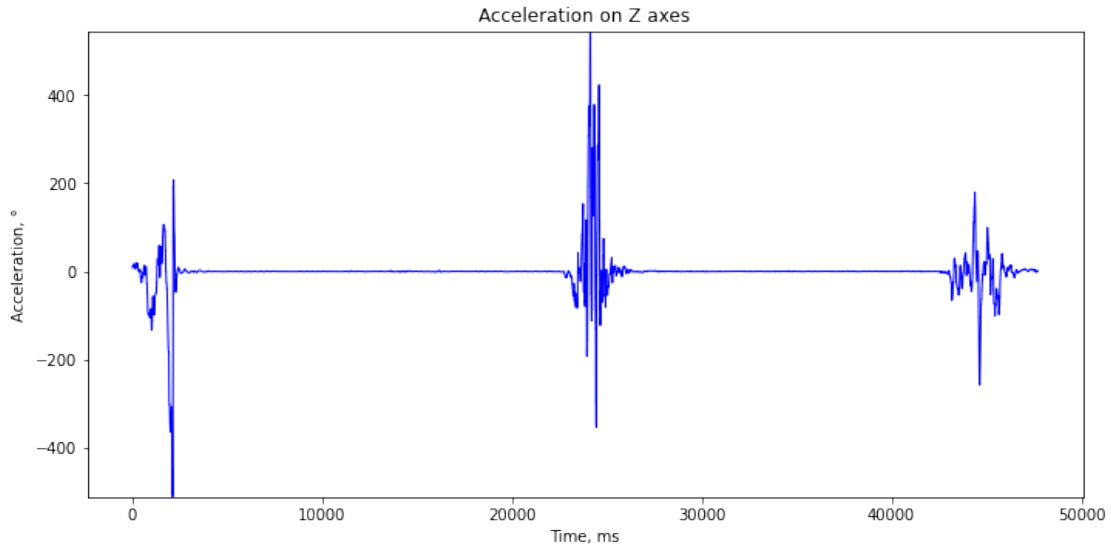

Acceleration on Z axes
Acceleration, °
300
200
100
0
−100
−200
−300
0
10000
20000
30000
40000
Time, ms

Run and Stay data

[40]:
```
_fig, ax3 = plt.subplots(nrows=3, ncols=1, constrained_layout=True,␣
↪figsize=(10,15))

plot_gyro(run_and_stay, ax3[0], 'x')
plot_gyro(run_and_stay, ax3[1], 'y')
plot_gyro(run_and_stay, ax3[2], 'z')

plt.show()
```


Acceleration on X axes
Acceleration, °/s
400
200
0
−200
−400
−600
0
10000
20000
30000
40000
50000
Time, ms


Acceleration on Y axes
Acceleration, °/s
750
500
250
0
−250
−500
−750
−1000
0
10000
20000
30000
40000
50000
Time, ms


Acceleration on Z axes
Acceleration, °
400
200
0
−200
−400
0
10000
20000
30000
40000
50000
Time, ms

1.2 Part 2: Sound signals

Source used: link

2 COULD NOT FINISH THE TASK. CAN NOT WORK WITH THE MICRO

[41]:
```
# import sounddevice as sd
# from scipy.io.wavfile import write
```

Set up recording configuration

[42]:
```
# Sampling frequency
# freq_8k = 8_000
# freq_44_1k = 44_100

# Recording duration
# duration = 5  # sec
```

Record the sounds

[43]:
```
# Start recorder with the given values of duration and sample frequency
# recording = sd.rec(int(duration * freq_44_1k), samplerate=freq_44_1k,␣
 ↪channels=1, blocking=True)

# Record audio for the given number of seconds
# sd.wait()
```

Save recordings to the File System

[44]:
```
# This will convert the NumPy array to an audio file with the given sampling␣
 ↪frequency
# write("recording0.wav", freq_8k, recording)
```

2.1 Part 3: EEG

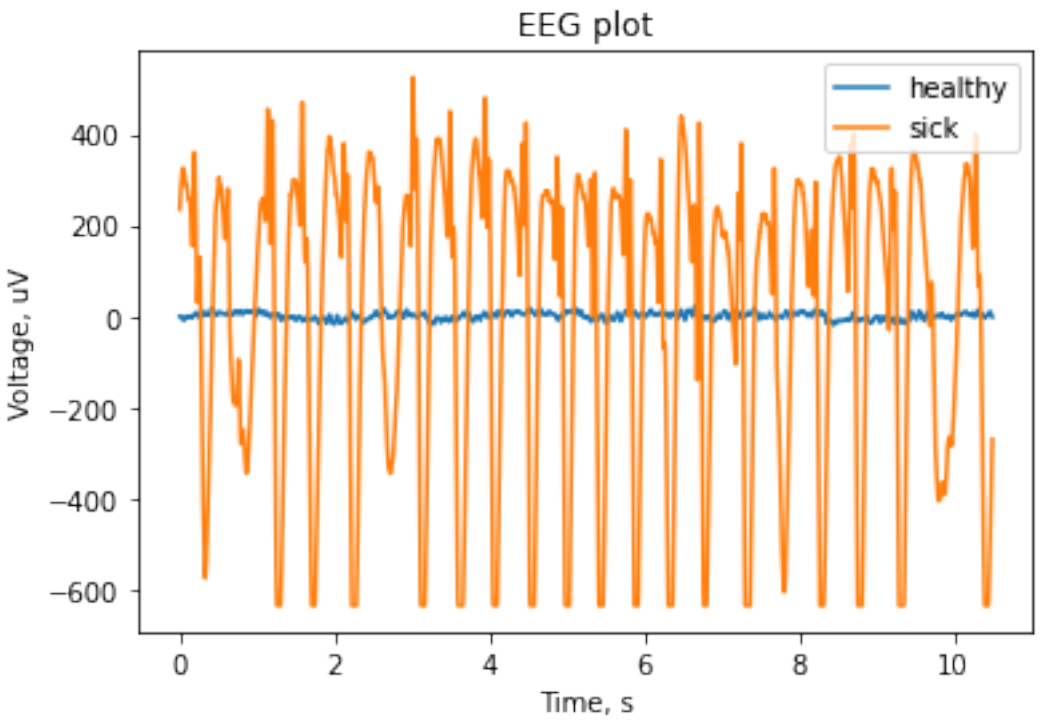
[2]:
```
def get_signal_time(signal: np.array, freq):
    return len(signal) / freq
```

[206]:
```
eeg_healthy = spio.loadmat('/home/fenix/pr/biosignal/lab_1_visualization/data/
 ↪eeg_healthy_15.mat', squeeze_me=True)['sig']
eeg_sick = spio.loadmat('/home/fenix/pr/biosignal/lab_1_visualization/data/
 ↪eeg_sick_15.mat', squeeze_me=True)['sig']
eeg_freq = 244
```

```
print(f"EEG of healthy person duration: {round(get_signal_time(eeg_healthy,␣
 ↪eeg_freq), 2)} s")
print(f"EEG of sick person duration:    {round(get_signal_time(eeg_sick,␣
 ↪eeg_freq), 2)} s")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-206-4951c03a9ed2> in <module>
      3 eeg_freq = 244
      4
----> 5 print(f"EEG of healthy person duration:␣
 ↪{round(get_signal_time(eeg_healthy, eeg_freq), 2)} s")
      6 print(f"EEG of sick person duration:    {round(get_signal_time(eeg_sick␣
 ↪eeg_freq), 2)} s")

NameError: name 'get_signal_time' is not defined
```

[47]:
```
t = np.linspace(0, get_signal_time(eeg_healthy, eeg_freq), len(eeg_healthy))
h_eeg_line = plt.plot(t, eeg_healthy, label='healthy')
s_eeg_line = plt.plot(t, eeg_sick, label='sick')
plt.title("EEG plot")
plt.xlabel('Time, s')
plt.ylabel('Voltage, uV')
plt.legend()
plt.show()
```



Save EEG to File

[8]:
```
import pickle

with open('../data/eeg_healthy.pyobj', 'wb+') as f_healthy:
    pickle.dump(eeg_healthy, f_healthy)
with open('../data/eeg_sick.pyobj', 'wb+') as f_healthy:
    pickle.dump(eeg_sick, f_healthy)

print("EEG py-data: stored")
```

```
EEG py-data: stored
```

2.2 Part 4: EKG

[4]:
```
from dataclasses import dataclass

@dataclass
class EKG_Data:
    fs: int
    units: str
    signal: np.ndarray
    labels: np.ndarray
    labels_indexes: np.ndarray
```

```
    source_start: np.ndarray
    source_end: np.ndarray
```

Load EKG dataset

[5]:
```
with open('/home/fenix/pr/biosignal/lab_1_visualization/data/norm_1600716798.
 ↪npz', 'rb') as f_norm:
    ekg_norm_npz = np.load(f_norm)
    ekg_norm = EKG_Data(
    ekg_norm_npz['fs'].item(0),
    ekg_norm_npz['units'].item(0),
    ekg_norm_npz['signal'],
    ekg_norm_npz['labels'],
    ekg_norm_npz['labels_indexes'],
    ekg_norm_npz['source_start'],
    ekg_norm_npz['source_end']
    )

with open('/home/fenix/pr/biosignal/lab_1_visualization/data/anomaly_1600718614.
 ↪npz', 'rb') as f_anomaly:
    ekg_anomaly_npz = np.load(f_anomaly)
    ekg_anomaly = EKG_Data(
    ekg_anomaly_npz['fs'].item(0),
    ekg_anomaly_npz['units'].item(0),
    ekg_anomaly_npz['signal'],
    ekg_anomaly_npz['labels'],
    ekg_anomaly_npz['labels_indexes'],
    ekg_anomaly_npz['source_start'],
    ekg_anomaly_npz['source_end']
    )

print("EKG data loaded")
```

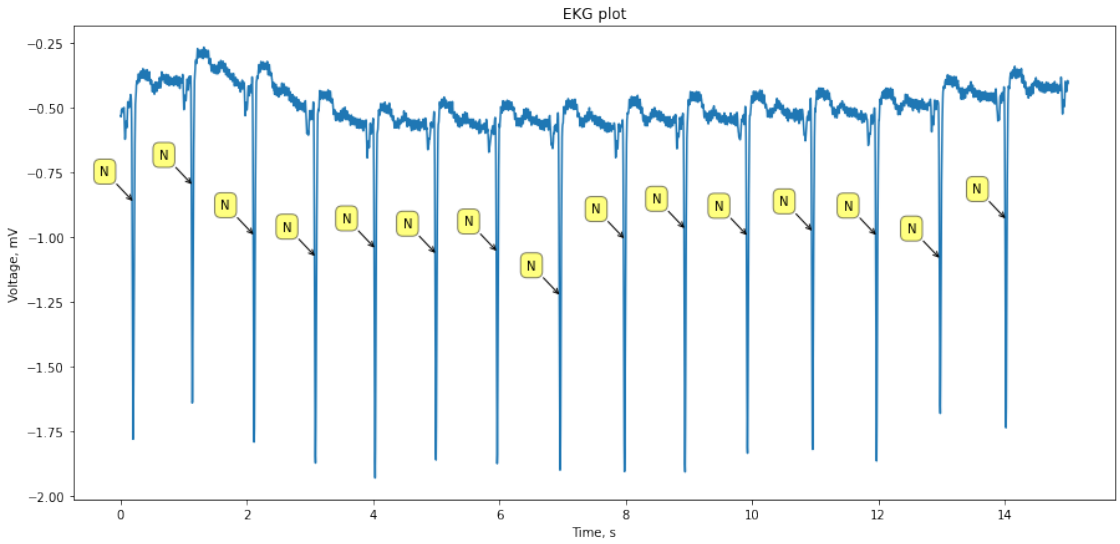
```
EKG data loaded
```

[14]:
```
def plot_ekg(data: EKG_Data):
    t = np.linspace(0, get_signal_time(data.signal, data.fs), data.signal.size)

    plt.figure(figsize=(15, 7))

    plt.plot(t, data.signal)
    plt.title("EKG plot")
    plt.xlabel('Time, s')
    plt.ylabel(f'Voltage, {data.units}')
    for label, x, y in zip(data.labels, t[data.labels_indexes], data.
 ↪signal[data.labels_indexes]):
        plt.annotate(
```

```
            label,
            xy=(x, y),
            xytext=(-20, 20),
            textcoords='offset points', ha='right', va='bottom',
            bbox=dict(boxstyle='round,pad=0.5', fc='yellow', alpha=0.5),
            arrowprops=dict(arrowstyle = '->', connectionstyle='arc3,rad=0')
        )
    plt.show()
```

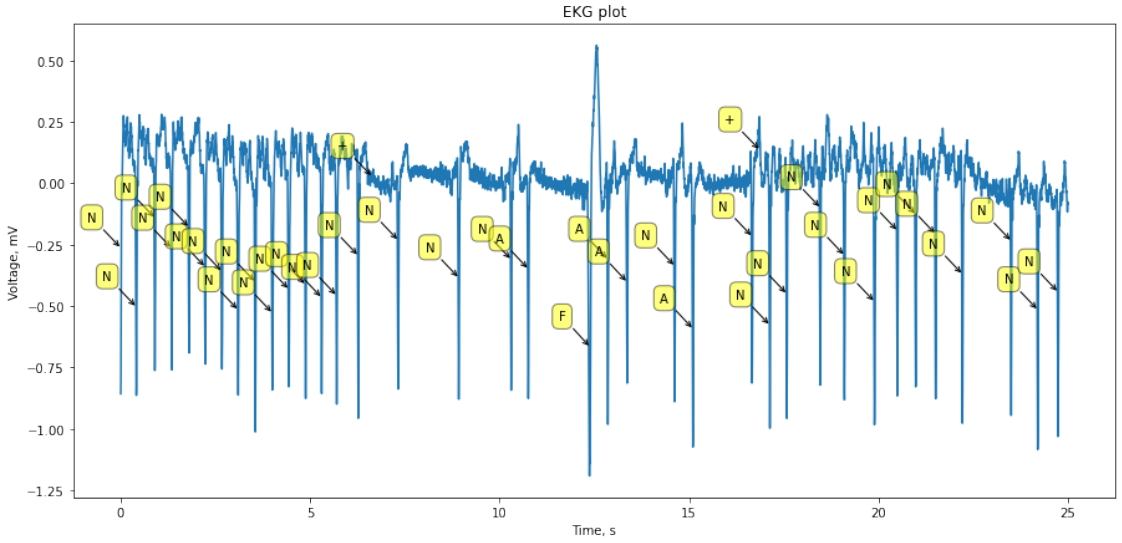
Plot normal EKG

[15]:
```
plot_ekg(ekg_norm)
```



Plot EKG with anomalies

[16]:
```
plot_ekg(ekg_anomaly)
```



2.3 Part 5: Cardiorhythmograms

2.3.1 Load Data

[25]:
```
hr_norm = spio.loadmat('/home/fenix/pr/biosignal/lab_1_visualization/data/
 ↪heart_rate_norm.mat', squeeze_me=True)['hr_norm']
hr_ap = spio.loadmat('/home/fenix/pr/biosignal/lab_1_visualization/data/
 ↪heart_rate_apnea.mat', squeeze_me=True)['hr_ap']

hr_norm
```

[25]:
```
array([   0,  872,  878, …, 1122, 1052, 1050], dtype=uint16)
```

[26]:
```
def get_hr_duration(data) -> int:
    return data.sum()
```

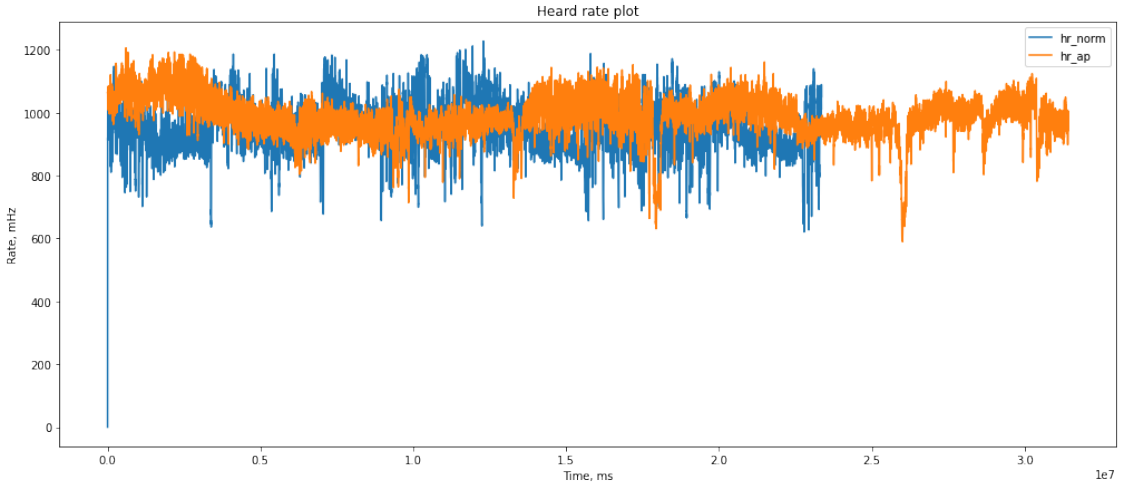
2.3.2 Signals durations

[27]:
```
print('Normal heart rate:', get_hr_duration(hr_norm), 'ms')
print('Apnea heart rate: ', get_hr_duration(hr_ap), 'ms')
```

```
Normal heart rate: 23354510 ms
Apnea heart rate:  31425568 ms
```

2.3.3 Interpolate and plot

[28]:
```
from scipy import interpolate
plt.figure(figsize=(17, 7))
def plot_hr(hr_data, label):
    t = np.cumsum(hr_data)
    f = interpolate.interp1d(t , hr_data)

    xnew = np.arange(hr_data.item(0), int(get_hr_duration(hr_data)), 1000)
    ynew = f(xnew)   # use interpolation function returned by `interp1d`

    plt.plot(xnew, ynew,'-', label=label)

    plt.title("Heard rate plot")
    plt.xlabel('Time, ms')
    plt.ylabel(f'Rate, mHz')

plot_hr(hr_norm, 'hr_norm')
plot_hr(hr_ap, 'hr_ap')
plt.legend()
plt.show()
```



2.4 Part 6: Stabilogram

[86]:
```
header = ['time_ms', 'top_left_f_kg', 'top_right_f_kg', 'bottom_left_f_kg',␣
 ↪'bottom_right_f_kg',
          'cop_x', 'cop_y', 'total_f']

base_close_acrob_files = [f'/home/fenix/pr/biosignal/lab_1_visualization/data/
 ↪acrobats/base_close/{i}.csv'
```

```
                          for i in range(1, 12)]
base_close_handb_files = [f'/home/fenix/pr/biosignal/lab_1_visualization/data/
 ↪handball/base_close/{i}.csv'
                          for i in range(1, 12)]

base_open_acrob_files = [f'/home/fenix/pr/biosignal/lab_1_visualization/data/
 ↪acrobats/base_open/{i}.csv'
                          for i in range(1, 12)]
base_open_handb_files = [f'/home/fenix/pr/biosignal/lab_1_visualization/data/
 ↪handball/base_open/{i}.csv'
                          for i in range(1, 12)]

def read_stabilogram(f_path: str):
    data = pd.read_csv(f_path, delim_whitespace=True, header=None, names=header)
    data['time_ms'] = data['time_ms'] - data['time_ms'].iloc[0]
    return data

base_close_acrobats = read_stabilogram(base_close_acrob_files[0])
base_close_handball = read_stabilogram(base_close_handb_files[0])

base_open_acrobats = read_stabilogram(base_open_acrob_files[0])
base_open_handball = read_stabilogram(base_open_handb_files[0])

base_open_handball.head()
```

[86]:

	time_ms	top_left_f_kg	top_right_f_kg	bottom_left_f_kg	\
0	0	16.2337	18.7797	20.5536	
1	12	16.2337	18.7797	20.5536	
2	19	16.3041	18.7992	20.4551	
3	29	16.3041	18.8770	20.5142	
4	39	16.3041	18.8770	20.5142	

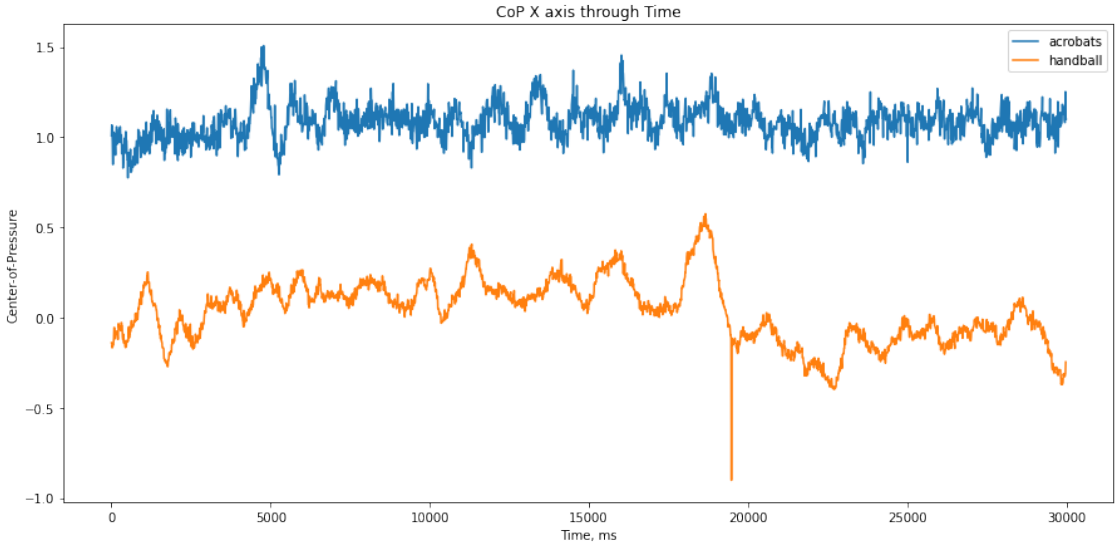
	bottom_right_f_kg	cop_x	cop_y	total_f
0	19.0848	0.360784	0.929303	74.6518
1	19.0848	0.360784	0.929303	74.6518
2	19.0946	0.379920	0.893401	74.6530
3	19.0848	0.382280	0.886166	74.7802
4	19.0848	0.382280	0.886166	74.7802

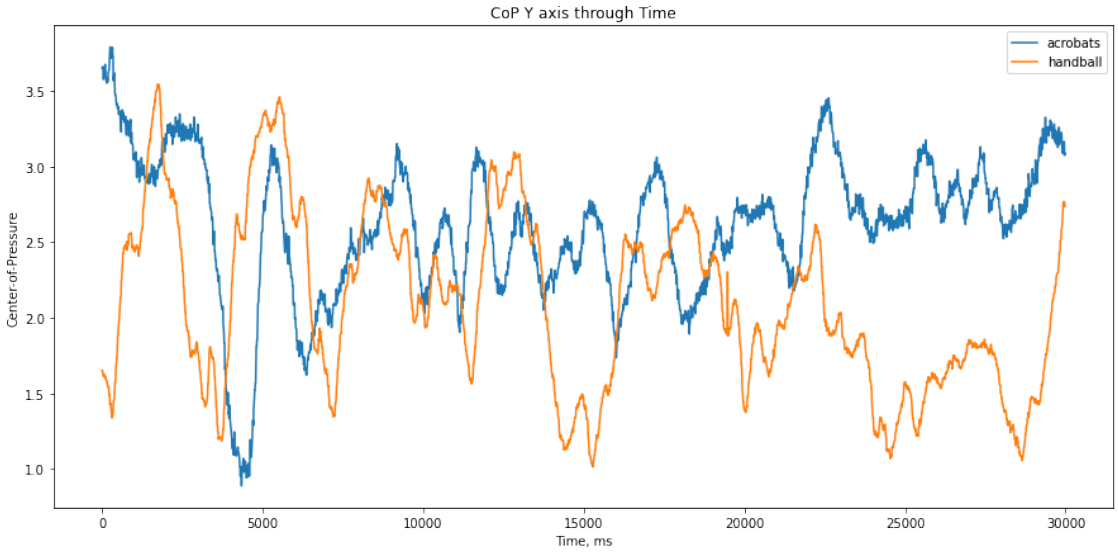
[60]:
```
def cop_plot(data_acrob, data_handb, axis: str):
    plt.figure(figsize=(15, 7))

    plt.plot(data_acrob['time_ms'], data_acrob[f'cop_{axis}'], label='acrobats')
    plt.plot(data_handb['time_ms'], data_handb[f'cop_{axis}'], label='handball')

    plt.title(f"CoP {axis.upper()} axis through Time")
    plt.xlabel('Time, ms')
```

```
    plt.ylabel('Center-of-Pressure')
    plt.legend()
    plt.show()
```

2.4.1 Base Close plot

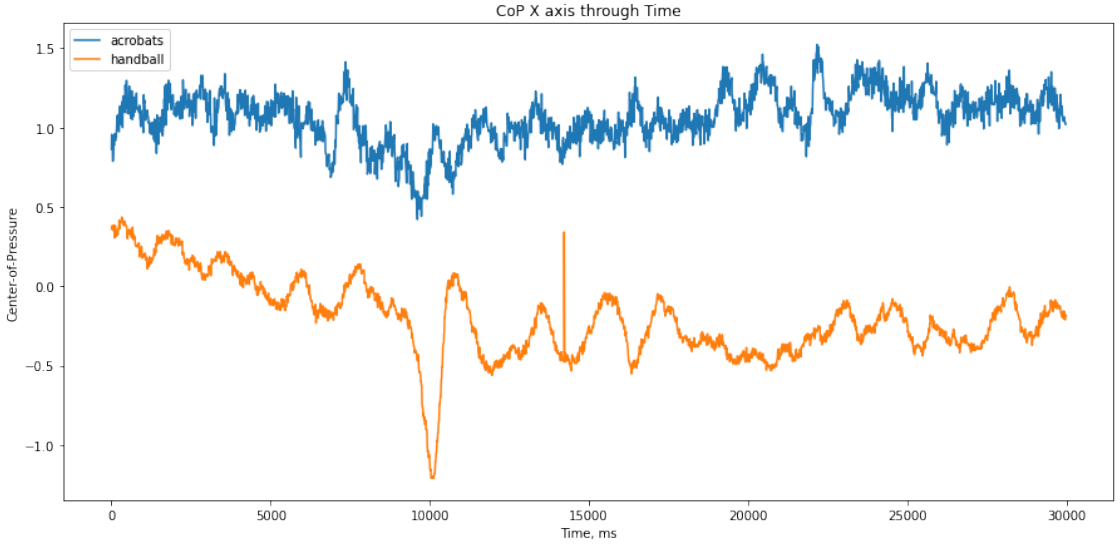
[62]:
```
cop_plot(base_close_acrobats, base_close_handball, 'x')
cop_plot(base_close_acrobats, base_close_handball, 'y')
```

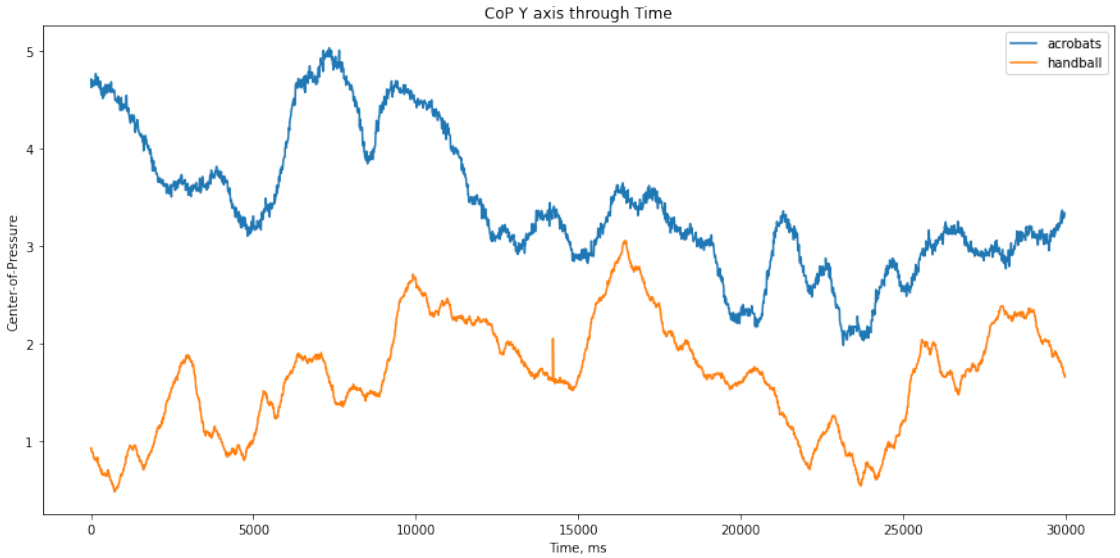




2.4.2 Base Ppen plot

[63]:
```
cop_plot(base_open_acrobats, base_open_handball, 'x')
cop_plot(base_open_acrobats, base_open_handball, 'y')
```





2.4.3 Select test to analyze

I whould like to choose tests **base_close** and **base_open**, to take into investigation the influence of eye view except coordination.

2.4.4 Statistics

[92]:
```
base_close_acrob_stat = pd.DataFrame()

stat_keys = ['mean_x', 'std_x', 'median_x', 'mean_y', 'std_y', 'median_y']
for f_path in base_close_acrob_files:
    el = read_stabilogram(f_path)
    tmp_stat = el.describe()

    data = pd.Series((tmp_stat['cop_x']['mean'], tmp_stat['cop_x']['std'], np.
 ↪median(el['cop_x']),
                tmp_stat['cop_y']['mean'], tmp_stat['cop_y']['std'], np.
 ↪median(el['cop_y'])),
               index=stat_keys)

    index = f_path.split('/')[-1]
    base_close_acrob_stat.insert(int(index.split('.')[0]) - 1, index, data,␣
 ↪True)

base_close_acrob_stat
```

[92]:

	1.csv	2.csv	3.csv	4.csv	5.csv	6.csv	\
mean_x	1.083326	-0.256687	-1.571683	-1.765780	-1.032887	-0.179734	
std_x	0.096335	0.349239	0.180992	0.157691	0.156439	0.201102	
median_x	1.079930	-0.189383	-1.550610	-1.775940	-1.053810	-0.181335	
mean_y	2.613995	3.881834	5.970359	2.522119	2.093707	5.880595	
std_y	0.465401	0.779344	0.384282	0.266361	0.492143	0.348774	
median_y	2.660290	3.746980	5.991450	2.528560	2.075760	5.836310	

	7.csv	8.csv	9.csv	10.csv	11.csv
mean_x	-1.127257	-0.141672	1.712125	-0.321524	2.179922
std_x	0.189810	0.314898	0.369186	0.240298	0.081607
median_x	-1.104955	-0.164476	1.750320	-0.284600	2.179290
mean_y	4.392782	4.908765	5.766000	1.947600	3.596429
std_y	0.529090	0.936724	0.763078	0.527665	0.246594
median_y	4.410065	5.000680	5.787950	1.949250	3.604970

[205]:
```
stat_keys = ['mean_x', 'std_x', 'median_x', 'mean_y', 'std_y', 'median_y']
def get_cop_stat(files):
    df = pd.DataFrame()

    for f_path in files:
        el = read_stabilogram(f_path)
        tmp_stat = el.describe()

        data = pd.Series((tmp_stat['cop_x']['mean'], tmp_stat['cop_x']['std'],␣
 ↪np.median(el['cop_x']),
```

```
                    tmp_stat['cop_y']['mean'], tmp_stat['cop_y']['std'], np.
↪median(el['cop_y'])),
                  index=stat_keys)

        index = f_path.split('/')[-1]
        df.insert(int(index.split('.')[0]) - 1, index, data, True)
    return df

def print_avg_stat(data, title):
    print(title.center(30))
    print("stat".ljust(10), "avg value")
    for i, key in enumerate(stat_keys):
        print(f"{key}:".ljust(10), f"{np.average(data.iloc[i])}")

base_close_acro_stat = get_cop_stat(base_close_acrob_files)
base_close_hand_stat = get_cop_stat(base_close_handb_files)
print()
print_avg_stat(base_close_acro_stat, "Base open handb stat")
print_avg_stat(base_close_hand_stat, "Base open handb stat")

base_open_acrob_stat = get_cop_stat(base_open_acrob_files)
base_open_handb_stat = get_cop_stat(base_open_handb_files)
print()
print_avg_stat(base_open_acrob_stat, "Base open handb stat")
print_avg_stat(base_open_handb_stat, "Base open handb stat")
```

```
     Base open handb stat
stat       avg value
mean_x:    -0.12925914477571282
std_x:     0.21250885823809976
median_x:  -0.11777895454545458
mean_y:    3.9612896556719766
std_y:     0.5217687648146696
median_y:  3.9629331818181823
     Base open handb stat
stat       avg value
mean_x:    0.030480575416047227
std_x:     0.17805507155174774
median_x:  0.02685813636363635
mean_y:    3.9593989611170772
std_y:     0.47937362069815487
median_y:  3.9443136363636366

     Base open handb stat
stat       avg value
mean_x:    -0.15085341919948236
```

```
std_x:     0.237296268001479
median_x:  -0.1562577
mean_y:    3.6757208750056622
std_y:     0.5835831634602507
median_y:  3.671434454545454
     Base open handb stat
stat       avg value
mean_x:    0.06925346708429116
std_x:     0.19764319422871848
median_x:  0.08034681818181819
mean_y:    4.3520254152722675
std_y:     0.4339313534326139
median_y:  4.346129090909091
```

Base close Statistics for Acrobats

[102]:
```
base_close_acro_stat
```

[102]:
```
              1.csv     2.csv     3.csv     4.csv     5.csv     6.csv  \
mean_x     1.083326 -0.256687 -1.571683 -1.765780 -1.032887 -0.179734
std_x      0.096335  0.349239  0.180992  0.157691  0.156439  0.201102
median_x   1.079930 -0.189383 -1.550610 -1.775940 -1.053810 -0.181335
mean_y     2.613995  3.881834  5.970359  2.522119  2.093707  5.880595
std_y      0.465401  0.779344  0.384282  0.266361  0.492143  0.348774
median_y   2.660290  3.746980  5.991450  2.528560  2.075760  5.836310

              7.csv     8.csv     9.csv    10.csv    11.csv
mean_x    -1.127257 -0.141672  1.712125 -0.321524  2.179922
std_x      0.189810  0.314898  0.369186  0.240298  0.081607
median_x  -1.104955 -0.164476  1.750320 -0.284600  2.179290
mean_y     4.392782  4.908765  5.766000  1.947600  3.596429
std_y      0.529090  0.936724  0.763078  0.527665  0.246594
median_y   4.410065  5.000680  5.787950  1.949250  3.604970
```

Base close Statistics for Handball Players

[103]:
```
base_close_hand_stat
```

[103]:
```
              1.csv     2.csv     3.csv     4.csv     5.csv     6.csv  \
mean_x     0.034958  0.805221 -0.568058 -0.293551 -0.214228  0.714325
std_x      0.171077  0.183777  0.299814  0.109648  0.216007  0.143171
median_x   0.054949  0.775682 -0.584033 -0.309114 -0.253682  0.709671
mean_y     2.100842  3.161899  4.558998  2.909131  4.554741  3.053592
std_y      0.579678  0.694346  0.634154  0.318815  0.499308  0.328736
median_y   2.052250  3.063425  4.584590  2.934970  4.564350  3.069745

              7.csv     8.csv     9.csv    10.csv    11.csv
mean_x     0.608252  0.504503  0.885641 -1.351371 -0.790405
```

```
std_x     0.099957  0.234944  0.235168  0.098676  0.166366
median_x  0.605807  0.524586  0.895852 -1.342110 -0.782168
mean_y    4.620612  3.383555  5.776892  5.645005  3.788122
std_y     0.288017  0.530082  0.584895  0.368221  0.446858
median_y  4.657340  3.314890  5.748570  5.636490  3.760830
```

Base open Statistics for Acrobats

[104]:
```
base_open_acrob_stat
```

[104]:
```
             1.csv     2.csv     3.csv     4.csv     5.csv     6.csv  \
mean_x    1.061045 -0.006626 -1.656090 -1.703635 -0.727730 -0.026144
std_x     0.162887  0.447216  0.144463  0.134056  0.119499  0.226913
median_x  1.069745 -0.072649 -1.660950 -1.696020 -0.725566 -0.034770
mean_y    3.422087  3.504612  5.706498  3.466846  1.184971  5.058763
std_y     0.713867  0.970805  0.357563  0.302419  0.383463  0.497785
median_y  3.241115  3.405900  5.755680  3.432160  1.244680  4.979830

             7.csv     8.csv     9.csv    10.csv    11.csv
mean_x   -2.860013 -0.208788  1.624314  0.776487  2.067791
std_x     0.185210  0.322720  0.363075  0.407889  0.096332
median_x -2.832725 -0.240204  1.570230  0.838904  2.065170
mean_y    4.700211  4.355016  5.507687 -0.074556  3.600795
std_y     0.452759  0.740788  0.741724  0.937152  0.321090
median_y  4.701565  4.459230  5.723780 -0.123921  3.565760
```

Base open Statistics for Handball Players

[105]:
```
base_open_handb_stat
```

[105]:
```
             1.csv     2.csv     3.csv     4.csv     5.csv     6.csv  \
mean_x   -0.198508  1.576975 -0.805688 -0.334458  0.152607  0.736969
std_x     0.248609  0.263585  0.324565  0.077534  0.160823  0.198122
median_x -0.210401  1.622270 -0.778459 -0.337358  0.157039  0.796831
mean_y    1.660781  3.449954  5.077961  4.339167  4.720584  3.239399
std_y     0.565903  0.497037  0.465610  0.238178  0.846217  0.299746
median_y  1.681190  3.507340  5.043580  4.308970  4.631290  3.274570

             7.csv     8.csv     9.csv    10.csv    11.csv
mean_x    0.859737  0.295599  0.669782 -0.998468 -1.192759
std_x     0.087861  0.122107  0.399798  0.170579  0.120493
median_x  0.865474  0.297453  0.663343 -0.989526 -1.202850
mean_y    4.540198  4.917243  6.491640  5.456119  3.979233
std_y     0.244236  0.586954  0.382619  0.356723  0.290021
median_y  4.551170  4.849260  6.519120  5.398930  4.042000
```

2.4.5 Summary

- Acrobates are more stable than handball players. This follows from the greater value of standard deviation for handball players. In simple words, handball players sway more.
- The median is the same as the mean. This is an indicator that the data tend to Normal distribution for both types of people.
- On axes X, the mean is almost 0 for all participants. But for axes Y, the value is bigger, which can tell that handball players are better prepared to start moving than acrobates. Logically, to start moving forward, we have to move our center of mass ahead to "fall" in that direction and then start to make steps.

2.5 Part 7: Heart Rate and SpO2

[151]:
```
df = pd.read_csv("/home/fenix/pr/biosignal/lab_1_visualization/data/
↪Subject15_SpO2Hr.csv")

# df.rename(columns=['Unnamed: 0', 'Elapsed time(seconds)', 'SpO2(%)', 'hr␣
↪(bpm)'], inplace = False)
df.rename(columns=dict(zip(['Unnamed: 0', 'Elapsed time(seconds)', 'SpO2(%)',␣
↪'hr (bpm)'],
                           ['0', 'time_s', 'SpO2_pers', 'hr_bpm'])),␣
↪inplace=True)
df.drop(['0'], axis='columns', inplace=True)

t = df['time_s'].to_numpy()
spo2 = df['SpO2_pers'].to_numpy()
hr_bpm = df['hr_bpm'].to_numpy()

df
```

[151]:

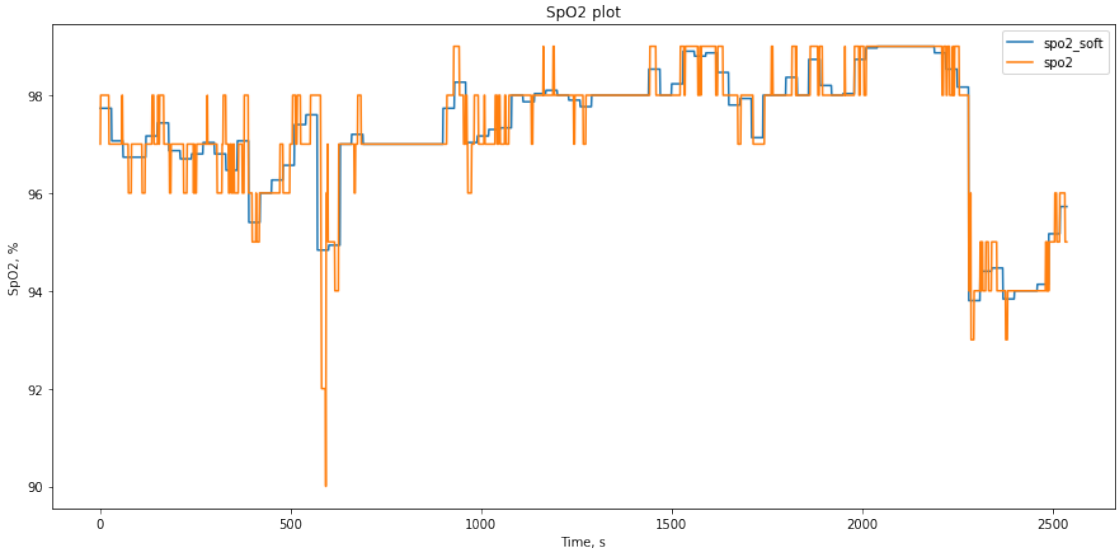
	time_s	SpO2_pers	hr_bpm
0	0.0	97.0	81.000
1	1.0	98.0	76.000
2	2.0	98.0	75.000
3	3.0	98.0	73.001
4	4.0	98.0	72.000
…	…	…	…
2533	2533.0	95.0	74.000
2534	2534.0	95.0	74.000
2535	2535.0	95.0	74.000
2536	2536.0	95.0	74.000
2537	2537.0	95.0	72.000

[2538 rows x 3 columns]

[155]:
```
from copy import deepcopy
plt.figure(figsize=(15, 7))
```

```
spo2_soft = deepcopy(spo2)
for i in range(0, len(spo2) + 30, 30):
    spo2_soft[i:i + 30] = np.average(spo2_soft[i:i + 30])

plt.plot(t, spo2_soft, label="spo2_soft")
plt.plot(t, spo2, label = "spo2")

plt.title("SpO2 plot")
plt.xlabel('Time, s')
plt.ylabel('SpO2, %')
plt.legend()
plt.show()
```

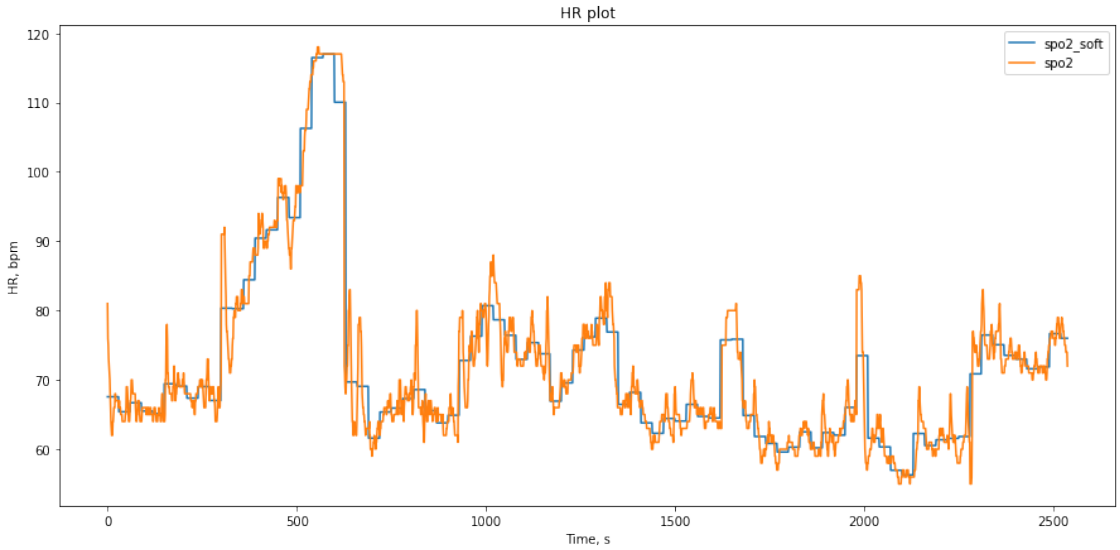
```
/home/fenix/pr/biosignal/lab_1_visualization/venv/lib/python3.8/site-
packages/numpy/lib/function_base.py:380: RuntimeWarning: Mean of empty slice.
  avg = a.mean(axis)
/home/fenix/pr/biosignal/lab_1_visualization/venv/lib/python3.8/site-
packages/numpy/core/_methods.py:170: RuntimeWarning: invalid value encountered
in double_scalars
  ret = ret.dtype.type(ret / rcount)
```



[156]:
```
plt.figure(figsize=(15, 7))
hr_bpm_soft = deepcopy(hr_bpm)
for i in range(0, len(hr_bpm) + 30, 30):
    hr_bpm_soft[i:i + 30] = np.average(hr_bpm_soft[i:i + 30])

plt.plot(t, hr_bpm_soft, label="spo2_soft")
plt.plot(t, hr_bpm, label = "spo2")
```

```
plt.title("HR plot")
plt.xlabel('Time, s')
plt.ylabel('HR, bpm')
plt.legend()
plt.show()
```

```
/home/fenix/pr/biosignal/lab_1_visualization/venv/lib/python3.8/site-
packages/numpy/lib/function_base.py:380: RuntimeWarning: Mean of empty slice.
  avg = a.mean(axis)
/home/fenix/pr/biosignal/lab_1_visualization/venv/lib/python3.8/site-
packages/numpy/core/_methods.py:170: RuntimeWarning: invalid value encountered
in double_scalars
  ret = ret.dtype.type(ret / rcount)
```



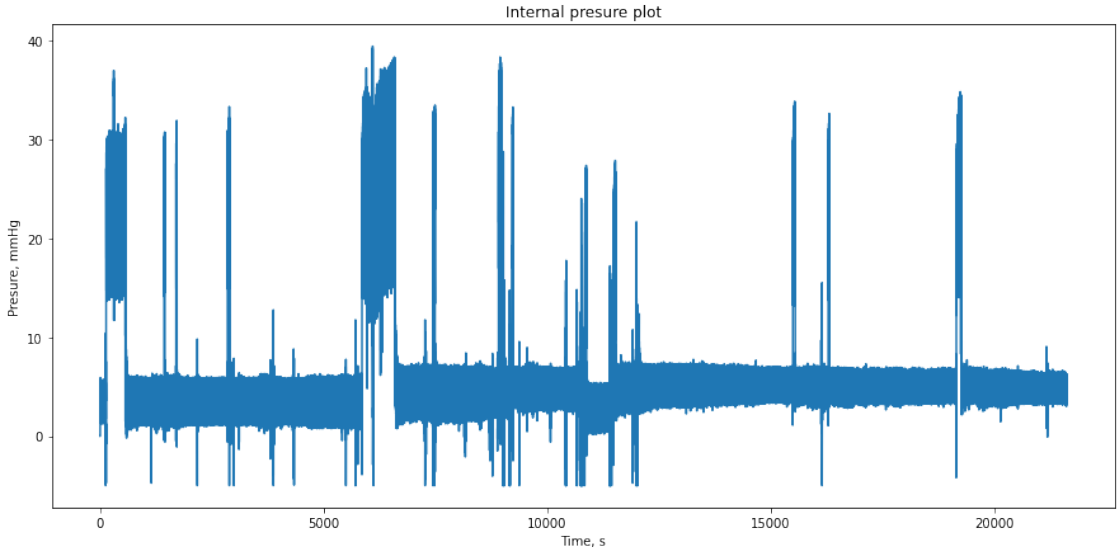
2.6 Part 8: Internl presure

[166]:
```
df = pd.read_csv('/home/fenix/pr/biosignal/lab_1_visualization/data/TBI_ICP.
 ↪txt',
                 delim_whitespace=True, header=None, names=['presure'])
presure = df['presure']
pr_freq = 125
pr_sign_time = len(presure) / 125
print(f"Total signal time: {pr_sign_time} s")
df
```

```
Total signal time: 21606.4 s
```

[166]:
```
         presure
0           0.00
1           0.00
2           0.00
3           0.00
4           0.00
…            …
2700795     3.80
2700796     3.65
2700797     3.55
2700798     3.55
2700799     3.60

[2700800 rows x 1 columns]
```

[168]:
```
plt.figure(figsize=(15, 7))
t = np.arange(0, pr_sign_time, 1 / pr_freq)
plt.plot(t, presure)

plt.title("Internal presure plot")
plt.xlabel('Time, s')
plt.ylabel('Presure, mmHg')

plt.show()
```



2.7 Part 9: Internl presure

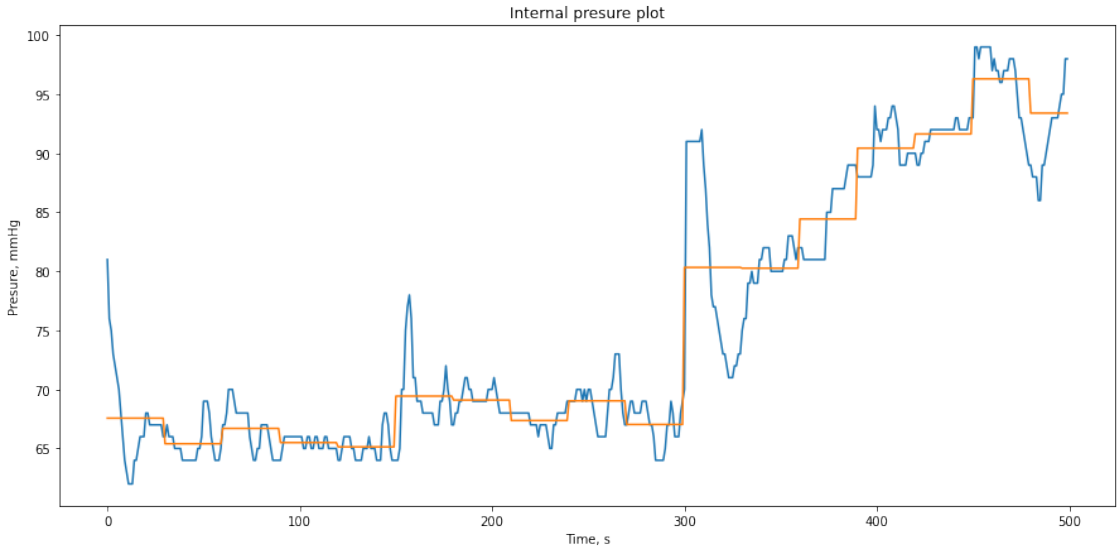
2.7.1 .

Input: - - ( ) - - .

Requirements: - -

.

[184]:
```
def plot_signal_segment(signal, start_t, end_t, freq, p_title='Signal plot',␣
↪p_signal_axes='signal',
                        p_size=(15, 7), get_segment=False, extert_plt=False,␣
↪create_plt=True):
    # Validate
    tatal_t = len(signal) / freq  # time in sec
    if start_t > end_t or end_t > tatal_t:
        raise RuntimeError("Invalid time range boundaries!")

    # Data construct
    start_i = int(start_t * freq)
    end_i = int(end_t * freq)
    signal_s = signal[start_i:end_i]
    t = np.arange(start_t, end_t, 1 / freq)

    # Plot
    if create_plt:
        plt.figure(figsize=p_size)

    plt.plot(t, signal_s)

    if create_plt:
        plt.title(p_title)
        plt.xlabel('Time, s')
        plt.ylabel(p_signal_axes)

    if not extert_plt:
        plt.show()

    if get_segment:
        return t, signal_s

plot_signal_segment(hr_bpm, 0, 500, 1, p_title='Internal presure plot',␣
↪p_signal_axes='Presure, mmHg', extert_plt=True)
plot_signal_segment(hr_bpm_soft, 0, 500, 1, create_plt=False)
```


Internal presure plot
Presure, mmHg
Time, s
100
95
90
85
80
75
70
65
0
100
200
300
400
500