DM - Mobility Task

Team members:

- Frick Bernhard (a01505541@unet.univie.ac.at)
- Postlmayr Billie Rosalie (a01307120@unet.univie.ac.at)

Former team members that opted out of the course:

- Decsi István (a11834026@unet.univie.ac.at)
- Krivanek Yvonne-Nadine (a01404589@unet.univie.ac.at)

Tokens:

- Frick: dm19_byrzma (id: 35)
- Postlmayr: dm19_postlmayr (id: 32)

In [1]:
```
%matplotlib inline
```

In [2]:
```
import os
import pandas as pd
import numpy as np
import sklearn
import matplotlib.pyplot as plt
import mplleaflet
import re
import dateutil.parser
from matplotlib.dates import date2num
from IPython.display import display, HTML
from datetime import datetime
```

In [3]:
```
data_dir = "data"
```

In [4]:
```
all_files = os.listdir(data_dir)
```

Removing non-whitelisted trips

To account for wrongly annotated trips, we used a whitelist where we listed all trips that are correctly annotated:

https://docs.google.com/spreadsheets/d/11Ta24L86uB1UiKwSa5of5MKfUIaoXxq53kGWQ8U-AyI/edit#gid=0
(https://docs.google.com/spreadsheets/d/11Ta24L86uB1UiKwSa5of5MKfUIaoXxq53kGWQ8U-AyI/edit#gid=0)

In [5]:
```
whitelist_ids = {
    5, 46, 74, 127, 128, 129, 131, 165, 99, 105,
    107, 108, 109, 152, 202, 203, 13, 28, 43, 52,
    145, 146, 22, 24, 26, 42, 102, 103, 147, 214,
    215, 19, 134, 137, 139, 142, 220, 222, 223, 224,
    26, 248, 246, 245, 241, 84, 93, 235, 236, 33,
    36, 37, 40, 49, 120, 113, 114, 115, 116, 117,
    218, 219, 226, 227, 240, 21, 32, 38, 70, 95,
    199, 78, 82, 83, 160, 161, 208, 209, 55, 9,
    35, 88, 210, 211, 217, 72, 47, 201, 204, 91,
    110, 228, 233, 234, 237
}
```

In [6]:
```
print("Number of whitelisted trips:", len(whitelist_ids))
```

```
Number of whitelisted trips: 94
```

In [7]:
```
def filter_trips(trip):
    # skip files that are not a trip
    if re.search("^\d+_\d+_\d{4}-\d{2}-\d{2}T\d{6}\.\d{1,3}$", trip) is None:
        return False

    # extract the trip id
    trip_id = int(trip.split('_')[1])

    # keep trips that are whitelisted
    if trip_id in whitelist_ids:
        return True

    # otherwise: skip the trip
    return False
```

In [8]:
```
whitelisted_trips = list(filter(filter_trips, all_files))
```

In [9]:
```
# dict: trip-id -> folder name
all_trips = dict(map(lambda trip: (int(trip.split('_')[1]), trip), whitelisted_trips))
```

1. Our Trips

The following script puts out data about the following data for a given list of trips:

- The trip ID
- The duration of the trip
- A table with all markers
- A plot with the acceleration data and markers
- A plot of the location data of the trip

To plot the location data, we used https://github.com/jwass/mplleaflet (https://github.com/jwass/mplleaflet), a wrapper for pyplot and openstreetmap.org.

In [10]:
```
def get_trip_folders(ids):
    return dict(filter(lambda elem: elem[0] in ids, all_trips.items()))
```

In [11]:
```
def trip_overview(trips):
    print("Number of trips: ", len(trips))

    trips = get_trip_folders(trips)

    for tid, trip in trips.items():

        display(HTML(f"<h1>Trip id {tid}</h1>".format(tid)))

        # read markers file
        marker_col_names = ["time", "key", "value", "mode", "longitude", "latitude", "col7", "col8", "col9"]
        markers = pd.read_csv(os.path.join("data", trip, "markers.csv"), sep=';', names=marker_col_names, ski
prows=1)

        # total duration
        start = dateutil.parser.parse(markers.loc[markers.index[0], 'time'])
        stop = dateutil.parser.parse(markers.loc[markers.index[-1], 'time'])
        print("Duration:", stop-start)

        # print markers table
        mode_changes = markers[markers["key"] == "CGT_MODE_CHANGED"]
        display(HTML(mode_changes.to_html()))

        # prepare acceleration data for plot
        acceleration = pd.read_csv(os.path.join("data", trip, "acceleration.csv"))
        acceleration['time'] = date2num(pd.to_datetime(acceleration['time']))
        acceleration['acc_norm'] = np.linalg.norm(acceleration[['x', 'y', 'z']].values, axis=1)
        acceleration = acceleration.drop(['x', 'y', 'z'], axis=1)

        # plot acceleration data
        plt.figure(figsize=(20,10))
        plt.grid(True)
        plt.plot_date(acceleration['time'], acceleration['acc_norm'], linewidth=1, color='black', linestyle=
'solid', marker='None')

        max_acc = np.max(acceleration['acc_norm'])

        # markers for acceleration data
        for index, row in mode_changes.iterrows():
            x = date2num(pd.to_datetime(row["time"]))
            # vertical line
            plt.axvline(x=x)
            # mode text
            plt.text(x=x, y=max_acc*.95, s=row["mode"], rotation=90, fontsize=16)
        plt.show()

        # plot map with location data
        positions = pd.read_csv(os.path.join("data", trip, "positions.csv"))
        positions = positions[positions["location_source"] == 1]
        positions = positions.filter(items=['longitude', 'latitude'])
        plt.figure(figsize=(20,8))
        plt.plot(positions['longitude'], positions['latitude'], 'r.', markersize=4)
        display(mplleaflet.display())
```

Bernhard Frick

In [12]:
```
frick_trip_ids = {113, 114, 115, 116, 117, 218, 219, 226, 227, 240}
```

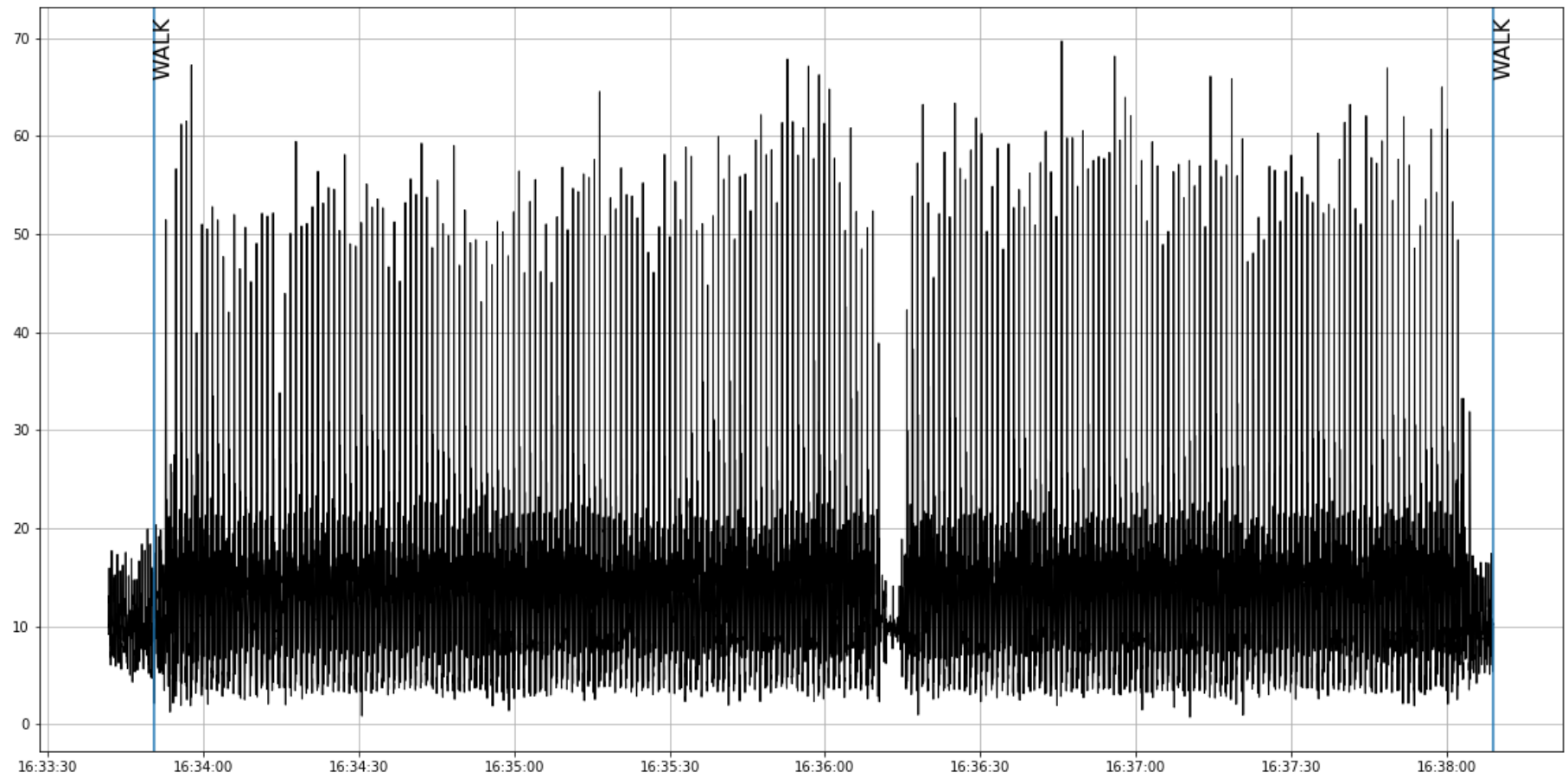
In [14]:
```
trip_overview(frick_trip_ids)
```

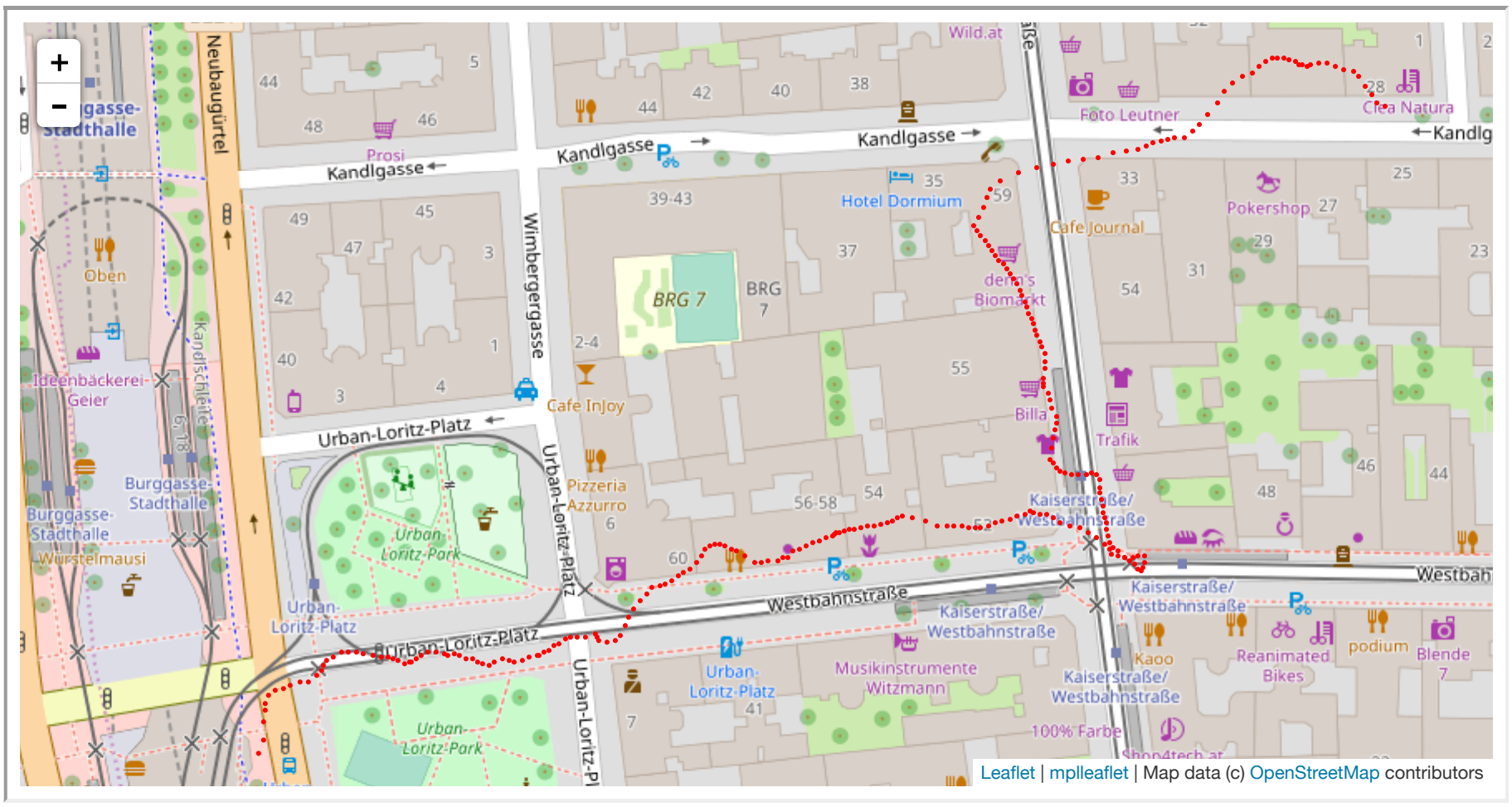
Number of trips: 10

Trip id 240

Duration: 0:04:27.430000

	time	key	value	mode	longitude	latitude	col7	col8	col9
4	2019-12-15T16:33:50.359Z	CGT_MODE_CHANGED	2019-12-15T16:33:41Z	WALK	16.33780542	48.20154978	-	Urban-Loritz-Platz	470227831.0
5	2019-12-15T16:38:09.031Z	CGT_MODE_CHANGED	2019-12-15T16:38:07Z	WALK	16.34142774	48.20284915	-	NaN	NaN

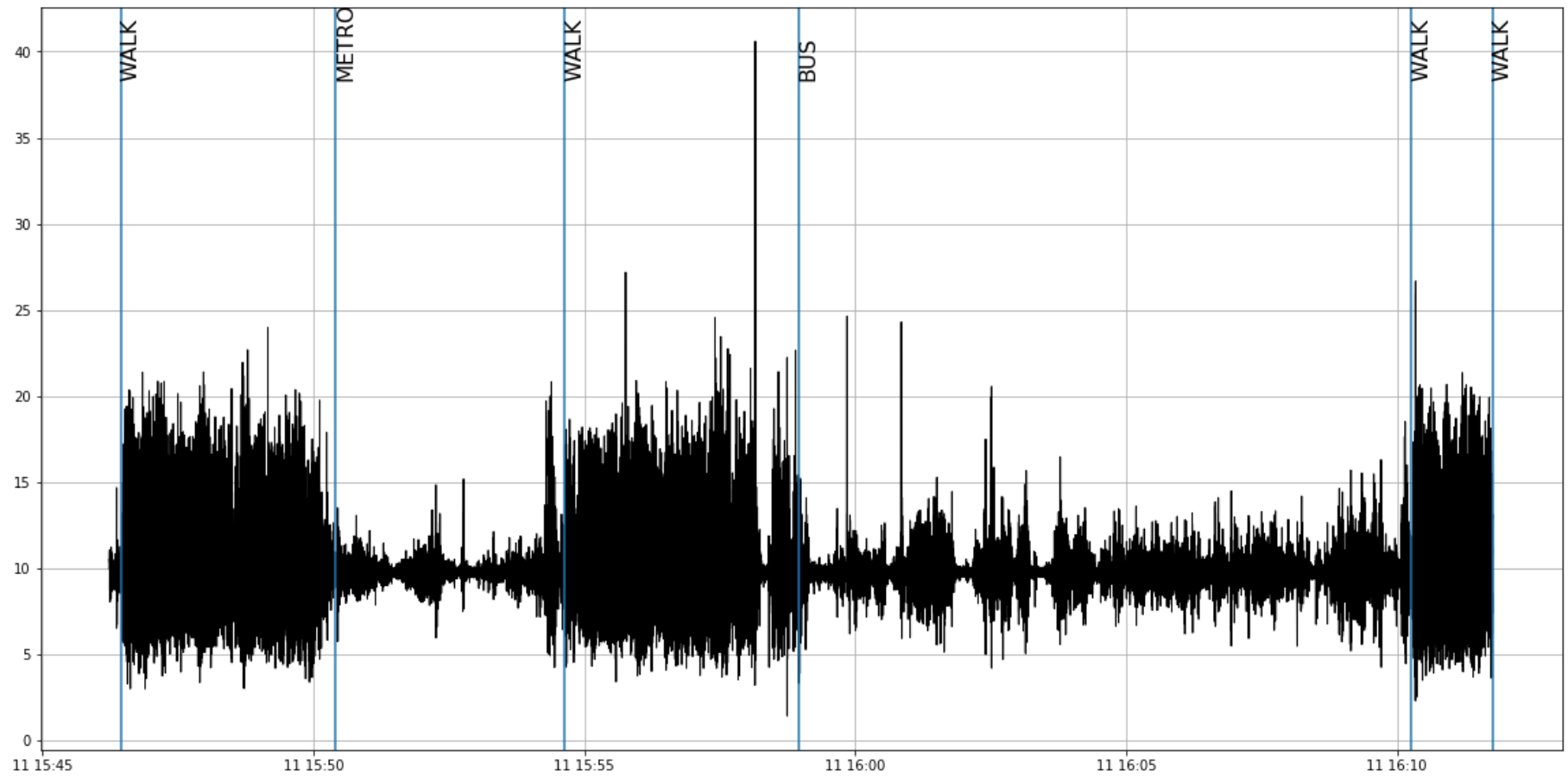


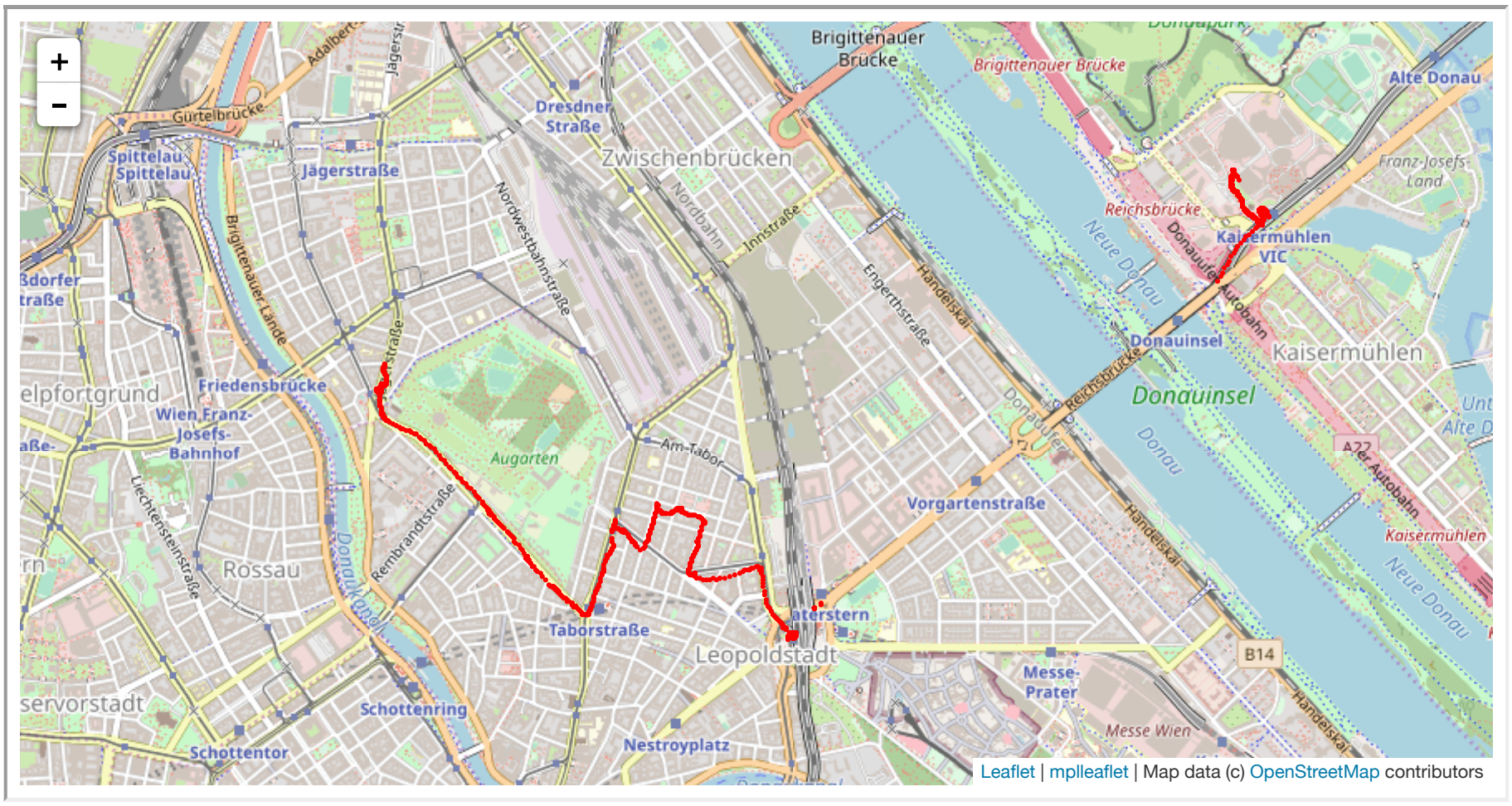


Trip id 115

Duration: 0:25:32.287000

	time	key	value	mode	longitude	latitude	col7	col8	col9
4	2019-11-11T15:46:26.711Z	CGT_MODE_CHANGED	2019-11-11T15:46:13Z	WALK	16.41436953	48.23426445	-	NaN	NaN
5	2019-11-11T15:50:24.373Z	CGT_MODE_CHANGED	2019-11-11T15:50:11Z	METRO	16.4161351	48.23280421	-	Kaisermühlen VIC	2.258520e+09
6	2019-11-11T15:54:38.247Z	CGT_MODE_CHANGED	2019-11-11T15:54:21Z	WALK	16.41350348	48.23057205	-	Praterstern	4.516847e+06
7	2019-11-11T15:58:56.991Z	CGT_MODE_CHANGED	2019-11-11T15:58:39Z	BUS	16.39144214	48.21840871	-	Praterstern	4.529408e+08
8	2019-11-11T16:10:15.959Z	CGT_MODE_CHANGED	2019-11-11T16:10:09Z	WALK	16.37018292	48.2267627	-	Gaußplatz	3.561490e+09
9	2019-11-11T16:11:46.024Z	CGT_MODE_CHANGED	2019-11-11T16:11:44Z	WALK	16.37042502	48.22771347	-	NaN	NaN

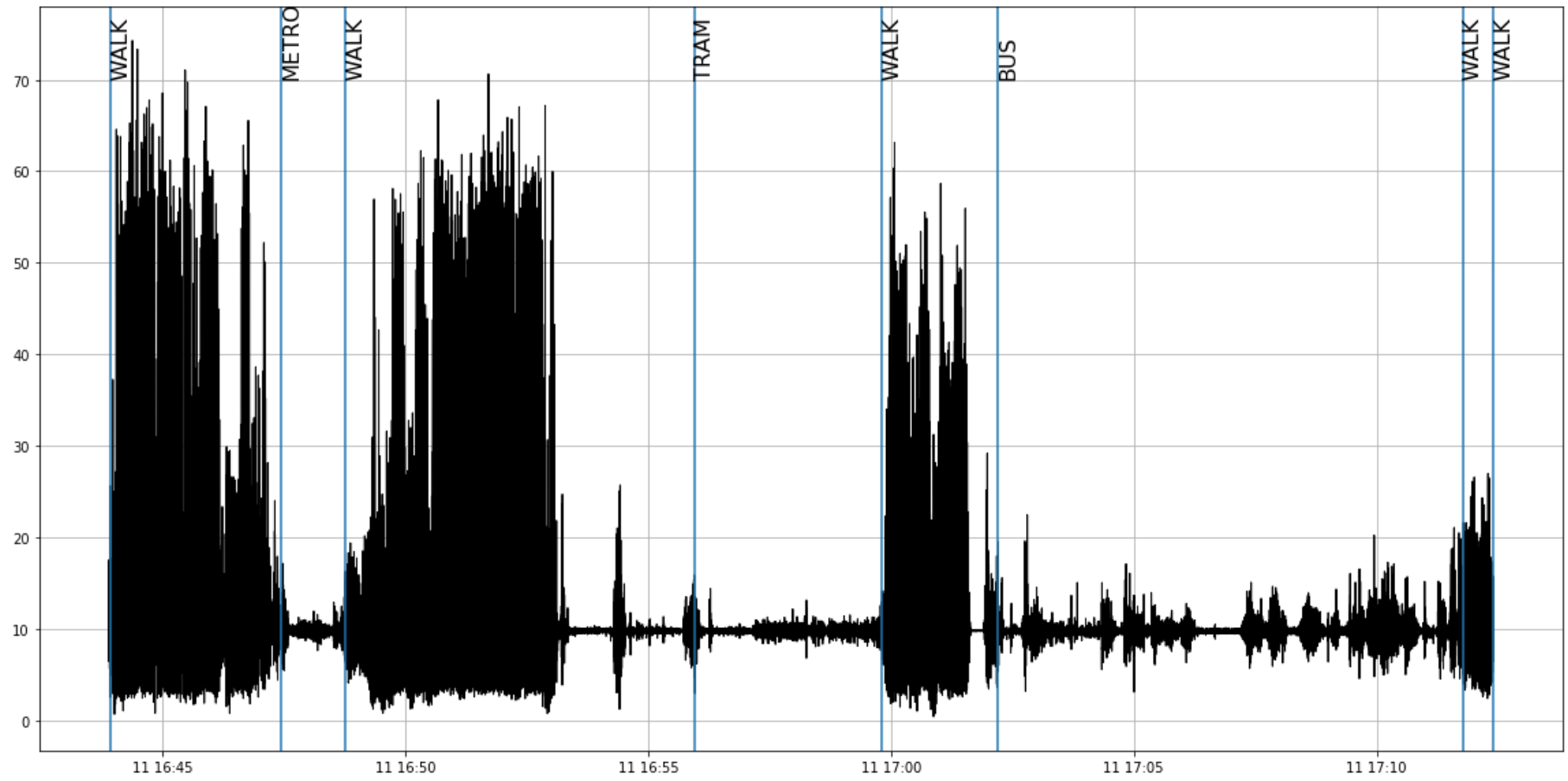


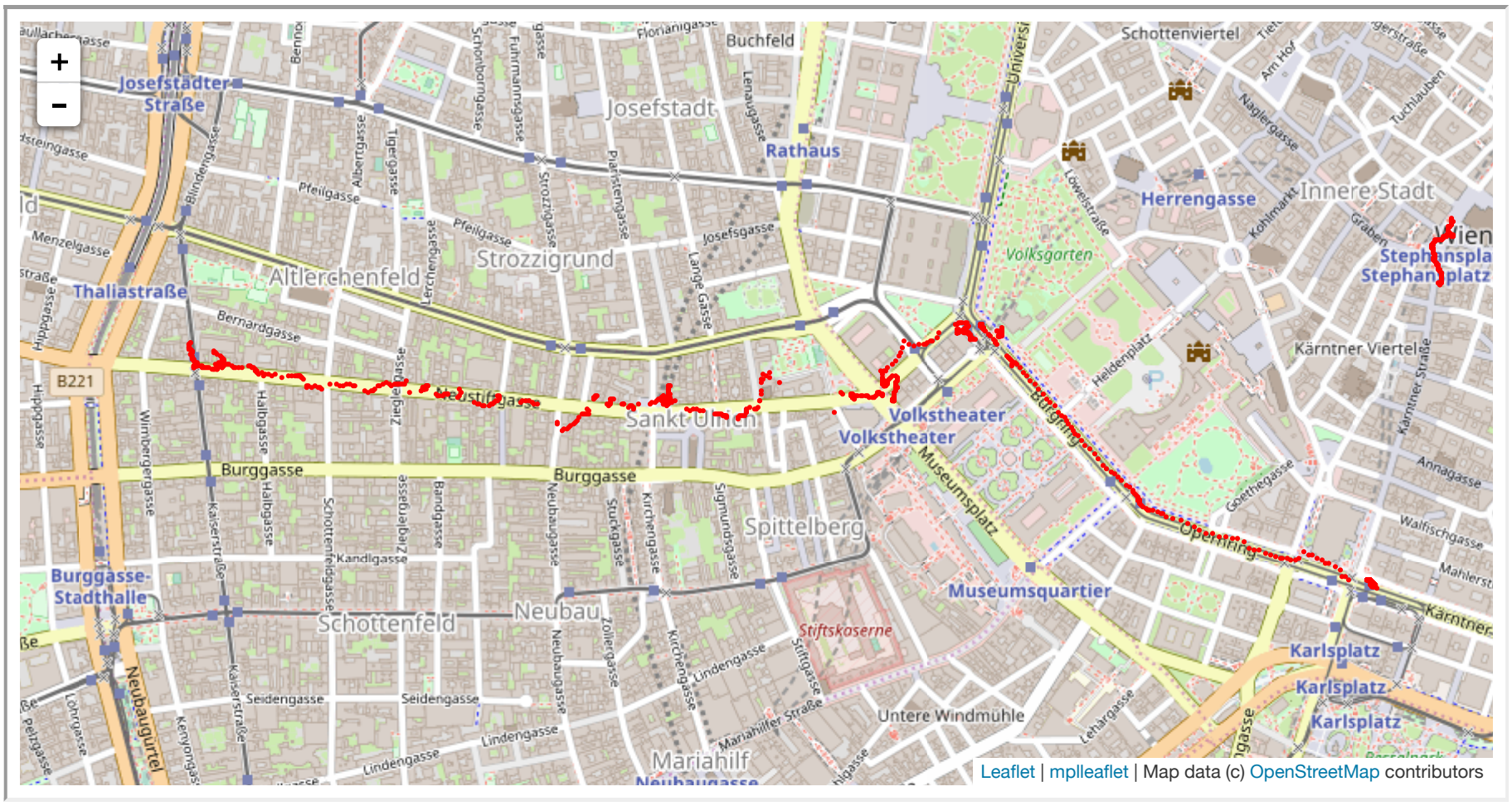


Trip id 116

Duration: 0:28:31.335000

	time	key	value	mode	longitude	latitude	col7	col8	col9
4	2019-11-11T16:43:55.450Z	CGT_MODE_CHANGED	2019-11-11T16:43:52Z	WALK	16.37245473	48.20872625	-	NaN	NaN
5	2019-11-11T16:47:26.484Z	CGT_MODE_CHANGED	2019-11-11T16:47:20Z	METRO	16.3719735	48.2082133	-	Stephansplatz	1.993538e+09
6	2019-11-11T16:48:45.624Z	CGT_MODE_CHANGED	2019-11-11T16:48:37Z	WALK	16.3719735	48.2082133	-	Karlsplatz	3.993343e+09
7	2019-11-11T16:55:56.702Z	CGT_MODE_CHANGED	2019-11-11T16:55:49Z	TRAM	16.37010495	48.20239925	-	Oper, Karlsplatz	5.592674e+08
8	2019-11-11T16:59:48.681Z	CGT_MODE_CHANGED	2019-11-11T16:59:44Z	WALK	16.35998919	48.20679458	-	Ring, Volkstheater	6.684652e+07
9	2019-11-11T17:02:11.772Z	CGT_MODE_CHANGED	2019-11-11T17:02:03Z	BUS	16.35961761	48.20677919	-	Ring, Volkstheater	6.067639e+07
10	2019-11-11T17:11:46.653Z	CGT_MODE_CHANGED	2019-11-11T17:11:43Z	WALK	16.33962612	48.20611093	-	Kaiserstraße/Neustiftgasse	7.475659e+06
11	2019-11-11T17:12:24.228Z	CGT_MODE_CHANGED	2019-11-11T17:12:22Z	WALK	16.33955284	48.20652193	-	NaN	NaN

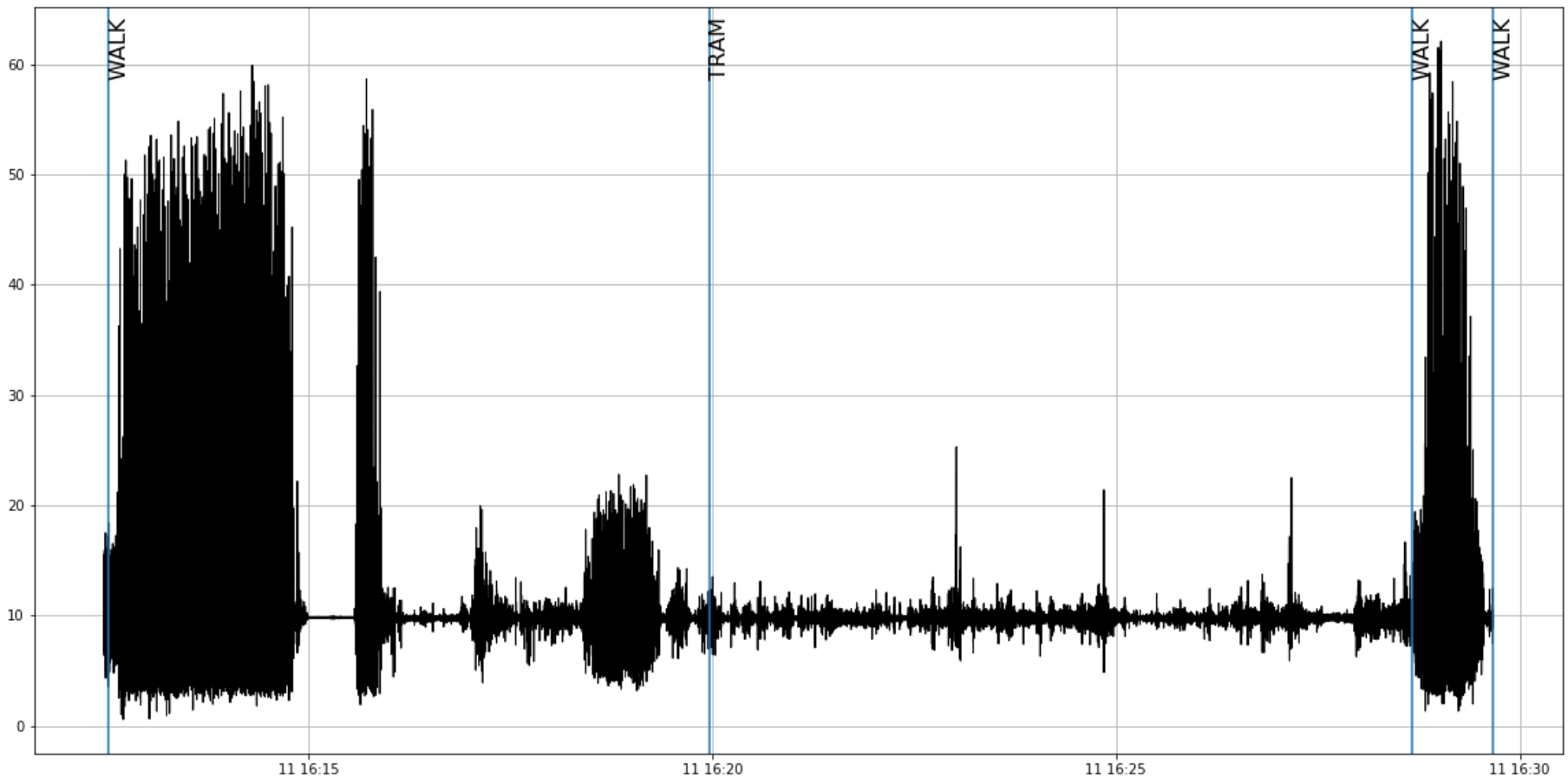


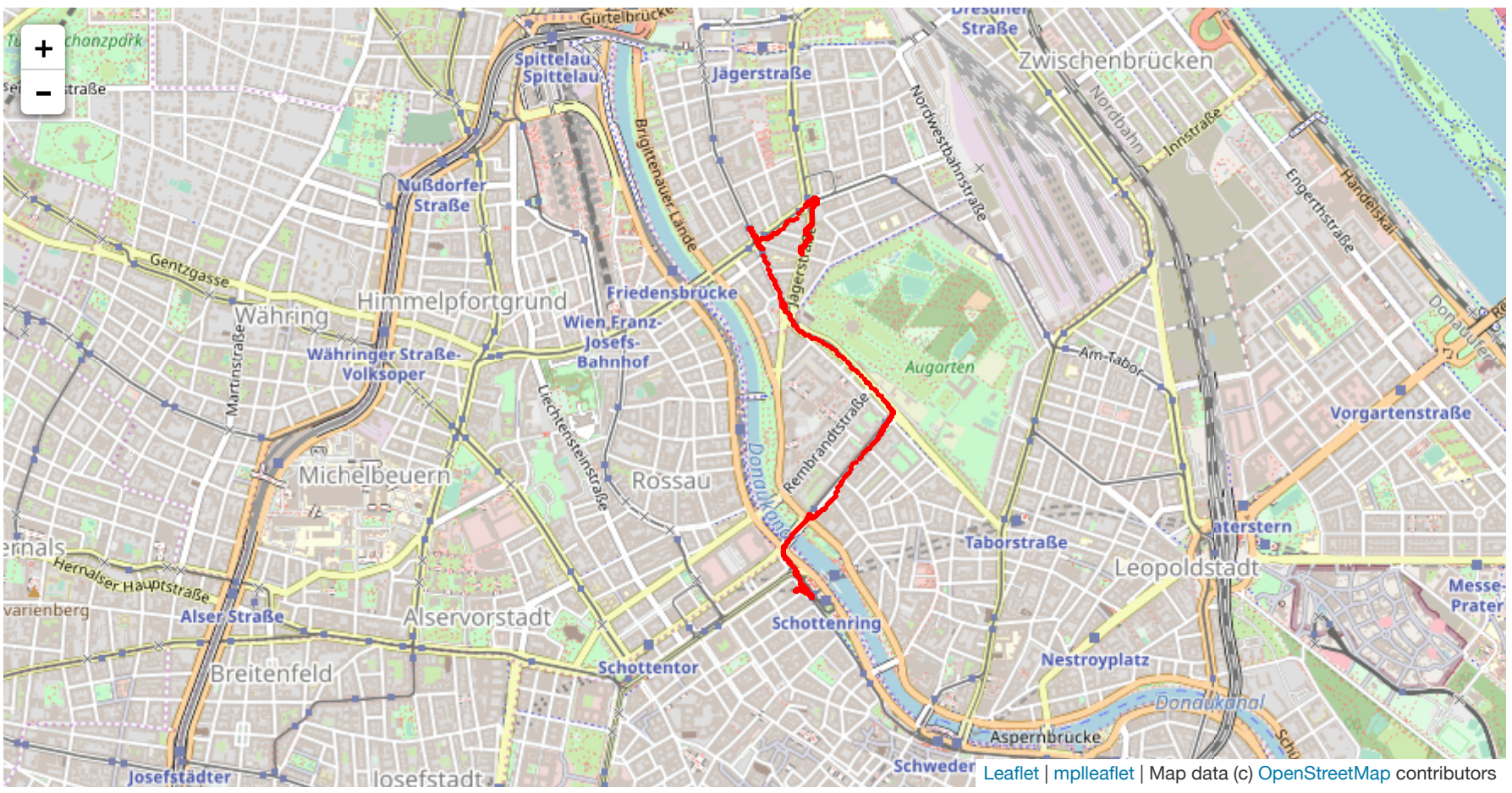


Trip id 114

Duration: 0:17:12.139000

	time	key	value	mode	longitude	latitude	col7	col8	col9
4	2019-11-11T16:12:31.912Z	CGT_MODE_CHANGED	2019-11-11T16:12:28Z	WALK	16.37067932	48.22829628	-	NaN	NaN
5	2019-11-11T16:19:58.489Z	CGT_MODE_CHANGED	2019-11-11T16:19:34Z	TRAM	16.36802111	48.22911409	-	Klosterneuburger Straße/Wallensteinstraße	2.590174e+09
6	2019-11-11T16:28:40.462Z	CGT_MODE_CHANGED	2019-11-11T16:28:34Z	WALK	16.37116945	48.21669471	-	Schottenring	6.684658e+07
7	2019-11-11T16:29:40.149Z	CGT_MODE_CHANGED	2019-11-11T16:29:38Z	WALK	16.37040384	48.21687698	-	NaN	NaN

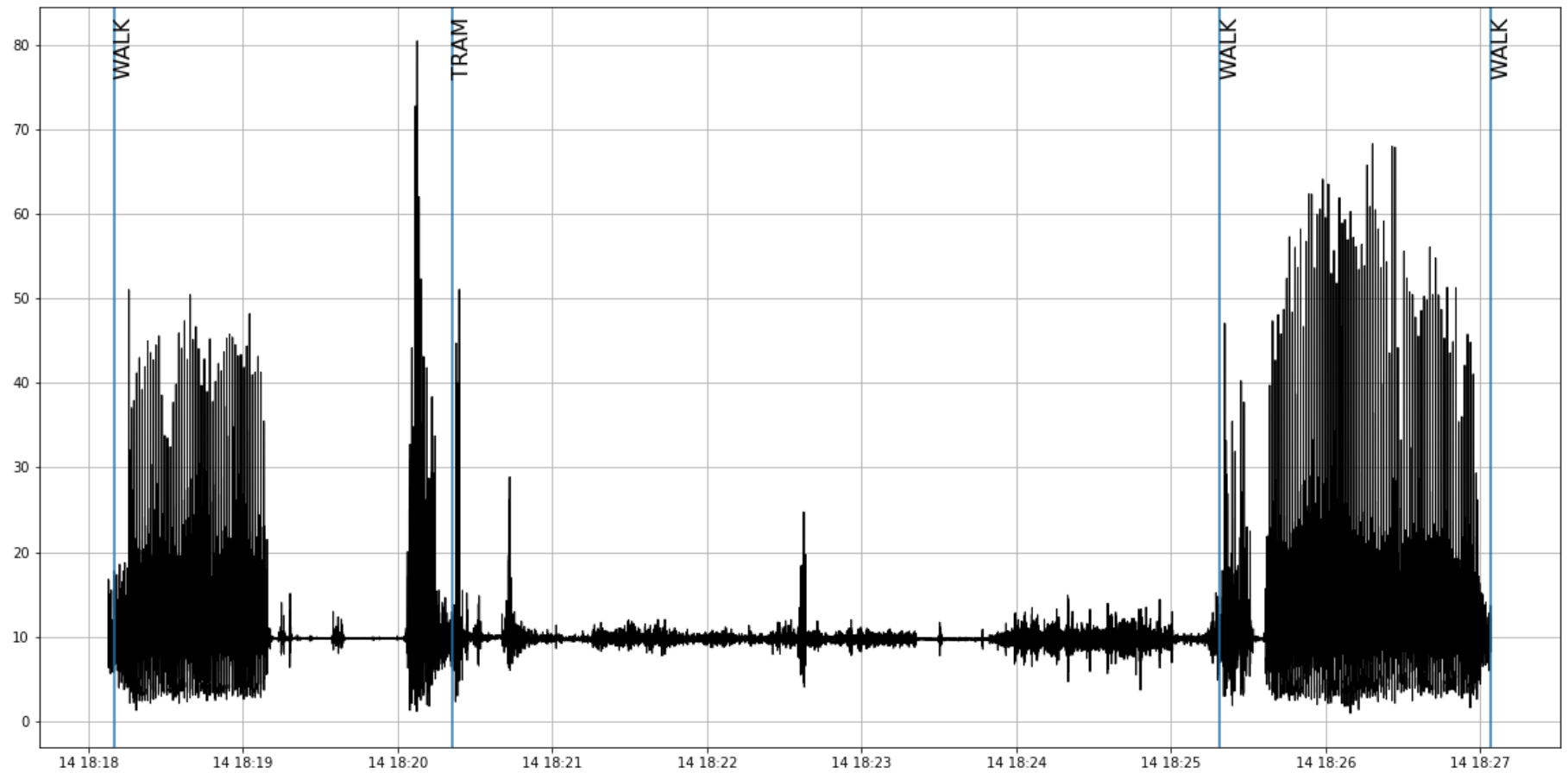


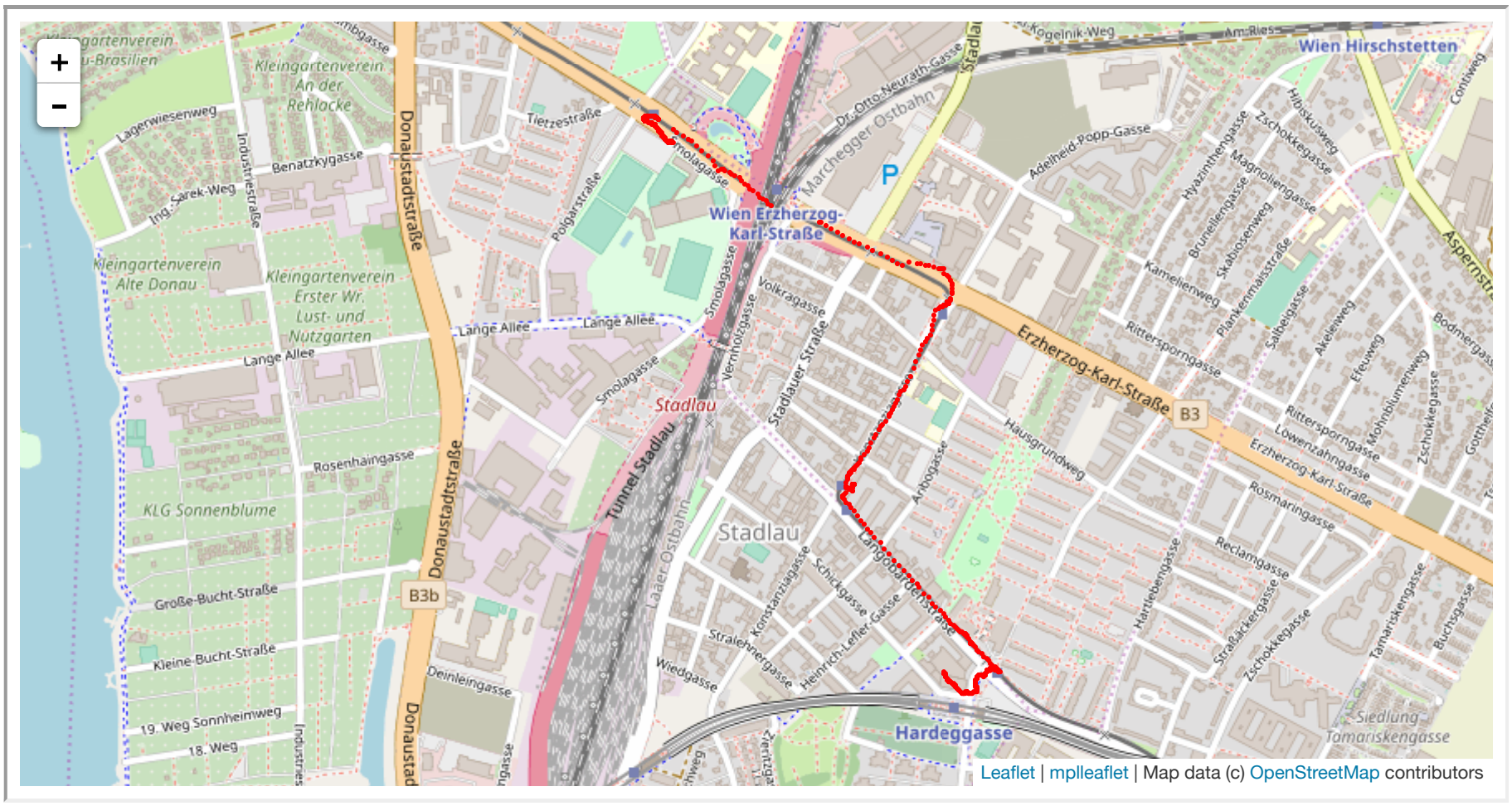


Trip id 226

Duration: 0:08:56.499000

	time	key	value	mode	longitude	latitude	col7	col8	col9
4	2019-12-14T18:18:10.022Z	CGT_MODE_CHANGED	2019-12-14T18:18:07Z	WALK	16.4510415	48.2300407	-	NaN	NaN
5	2019-12-14T18:20:21.172Z	CGT_MODE_CHANGED	2019-12-14T18:20:17Z	TRAM	16.45008725	48.23107328	-	Polgarstraße	1.595860e+09
6	2019-12-14T18:25:18.938Z	CGT_MODE_CHANGED	2019-12-14T18:24:51Z	WALK	16.45893122	48.22154113	-	Hardeggasse	6.855322e+08
7	2019-12-14T18:27:04.196Z	CGT_MODE_CHANGED	2019-12-14T18:27:03Z	WALK	16.45753141	48.22151964	-	NaN	NaN

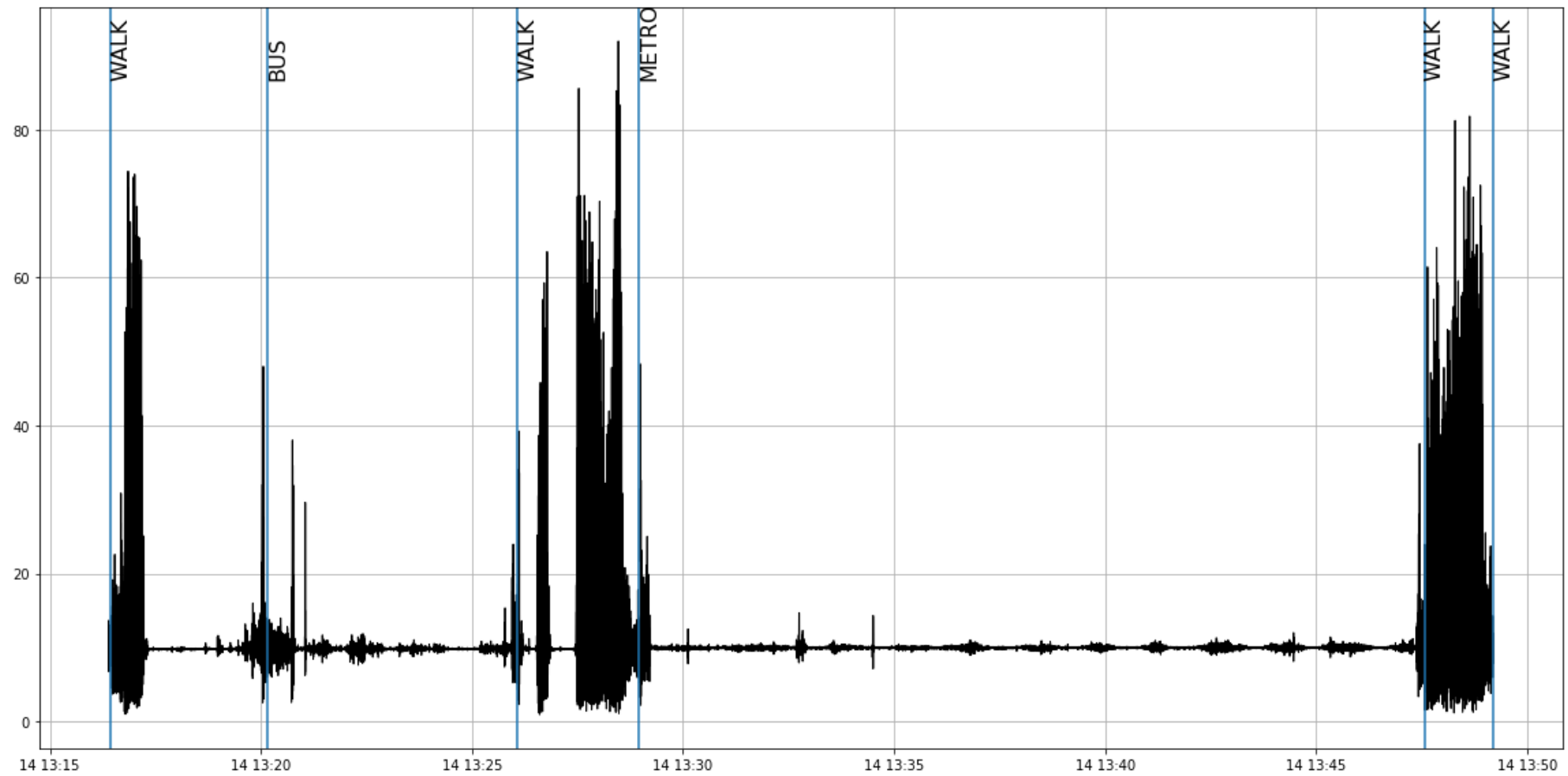


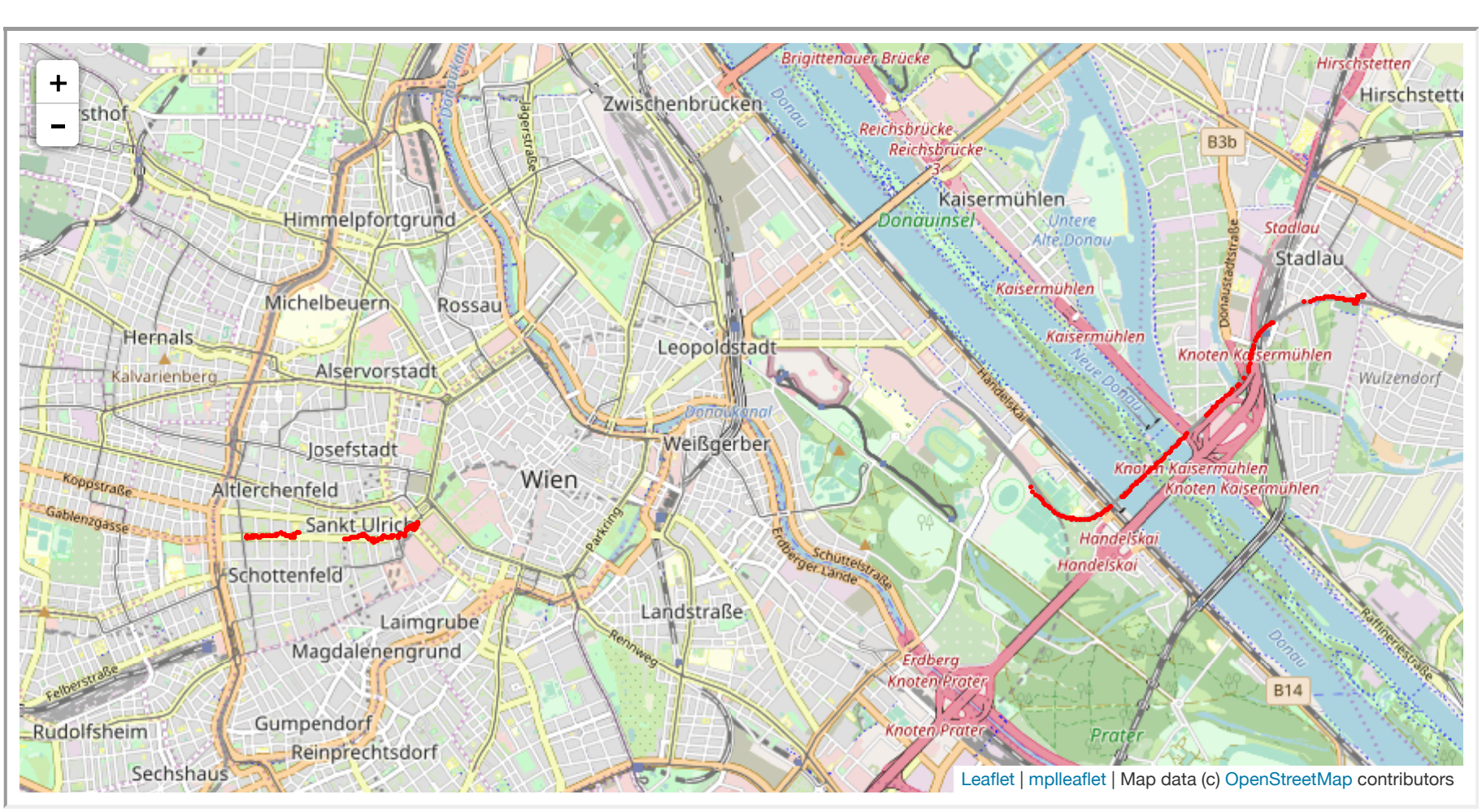


Trip id 219

Duration: 0:32:48.607000

	time	key	value	mode	longitude	latitude	col7	col8	col9
4	2019-12-14T13:16:26.847Z	CGT_MODE_CHANGED	2019-12-14T13:16:23Z	WALK	16.3412826	48.20436114	-	NaN	NaN
5	2019-12-14T13:20:09.927Z	CGT_MODE_CHANGED	2019-12-14T13:20:05Z	BUS	16.34043534	48.20435956	-	Kaiserstraße/Burggasse	3.581237e+09
6	2019-12-14T13:26:04.641Z	CGT_MODE_CHANGED	2019-12-14T13:26:01Z	WALK	16.35773388	48.20505529	-	Volkstheater	2.761714e+08
7	2019-12-14T13:28:57.123Z	CGT_MODE_CHANGED	2019-12-14T13:28:45Z	METRO	16.35810721	48.20496058	-	Volkstheater	2.476815e+09
8	2019-12-14T13:47:33.965Z	CGT_MODE_CHANGED	2019-12-14T13:47:29Z	WALK	16.4574412	48.22083472	-	Hardeggasse	3.530478e+09
9	2019-12-14T13:49:12.025Z	CGT_MODE_CHANGED	2019-12-14T13:49:10Z	WALK	16.45836779	48.22130711	-	NaN	NaN

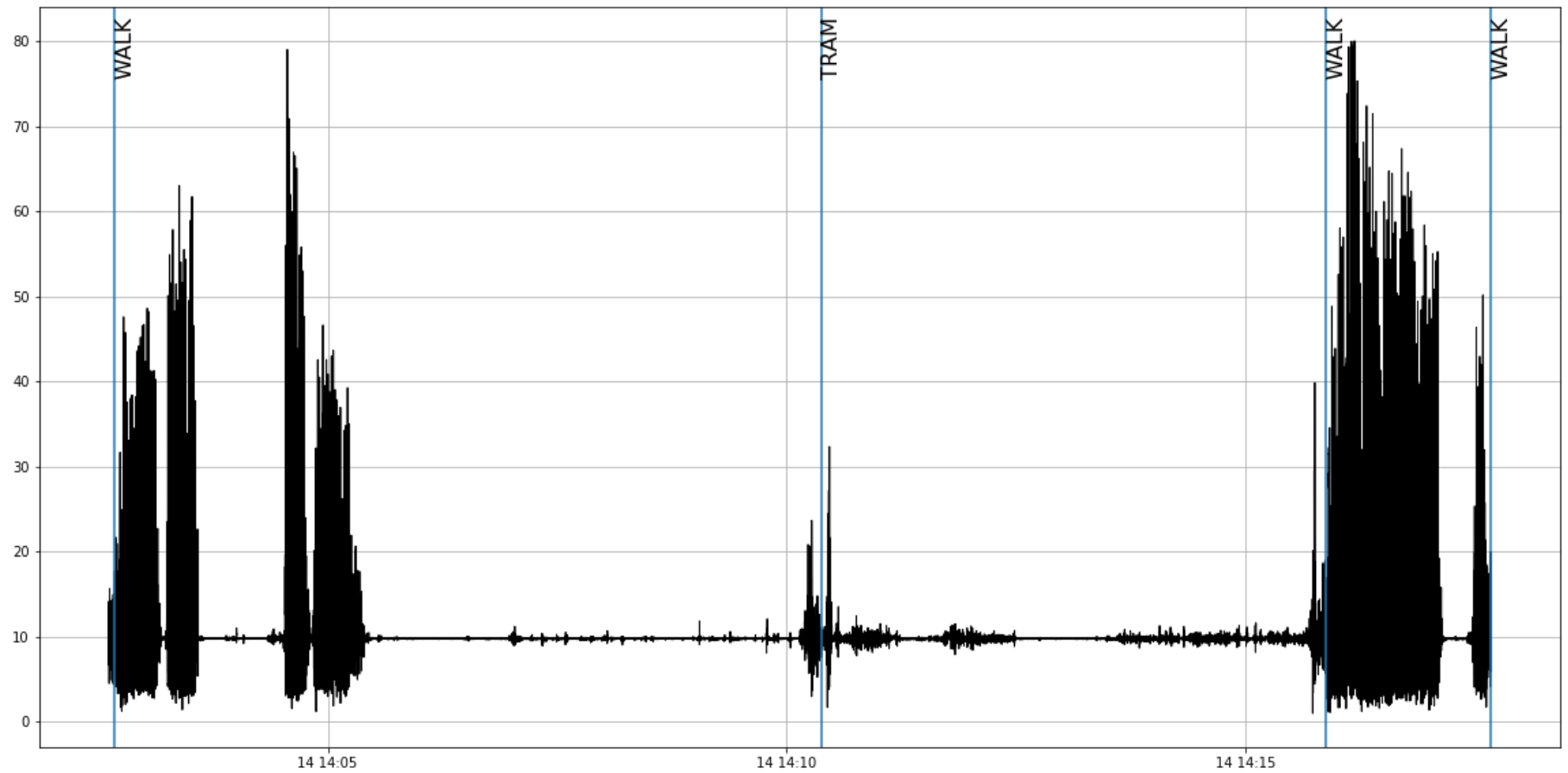


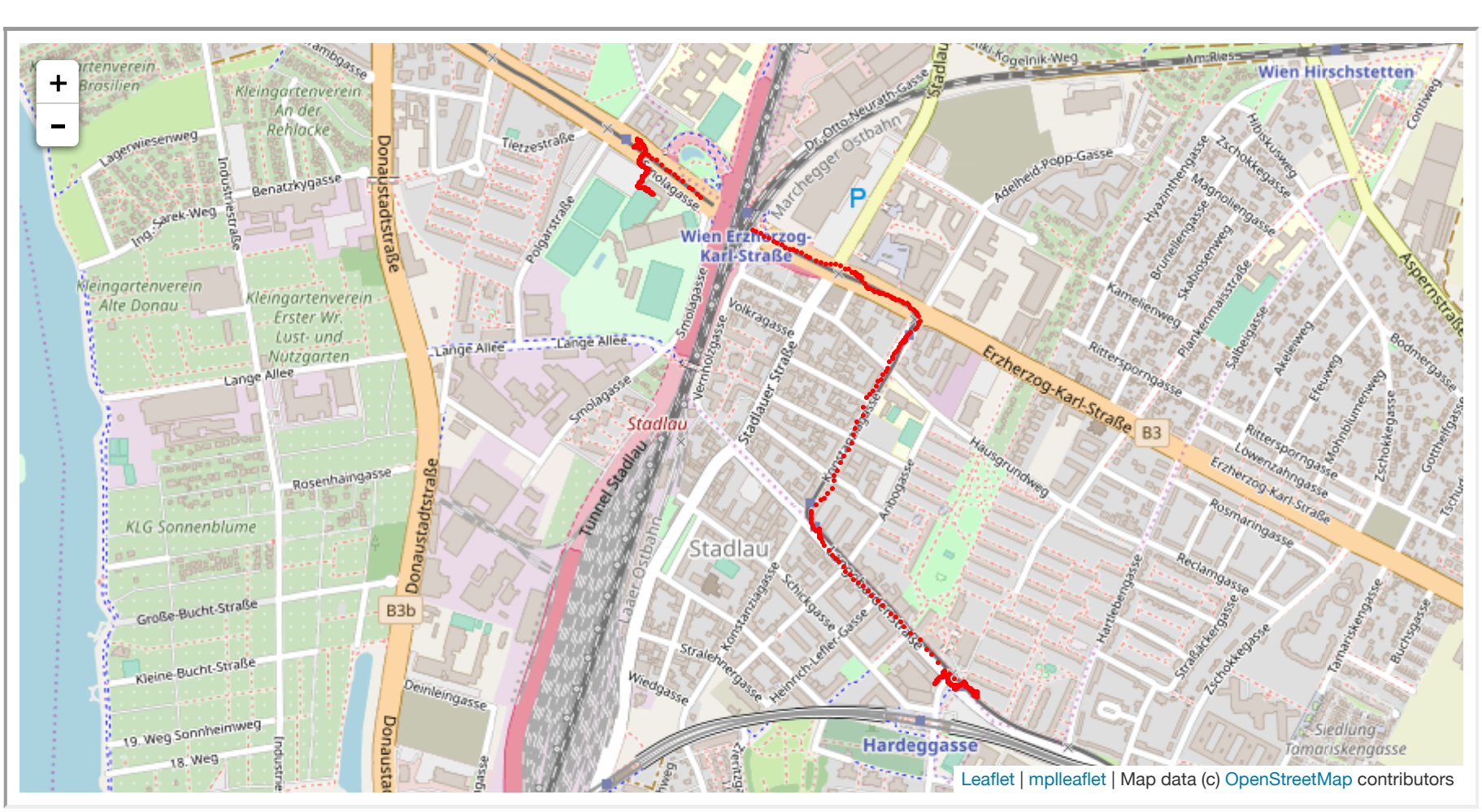


Trip id 218

Duration: 0:15:03.913000

	time	key	value	mode	longitude	latitude	col7	col8	col9
4	2019-12-14T14:02:40.654Z	CGT_MODE_CHANGED	2019-12-14T14:02:36Z	WALK	16.4579091	48.2213518	-	NaN	NaN
5	2019-12-14T14:10:23.289Z	CGT_MODE_CHANGED	2019-12-14T14:10:18Z	TRAM	16.45926485	48.22133726	-	Hardeggasse	1.372925e+09
6	2019-12-14T14:15:52.709Z	CGT_MODE_CHANGED	2019-12-14T14:15:47Z	WALK	16.45032575	48.23110618	-	Polgarstraße	1.595861e+09
7	2019-12-14T14:17:40.421Z	CGT_MODE_CHANGED	2019-12-14T14:17:39Z	WALK	16.45072654	48.23016849	-	NaN	NaN

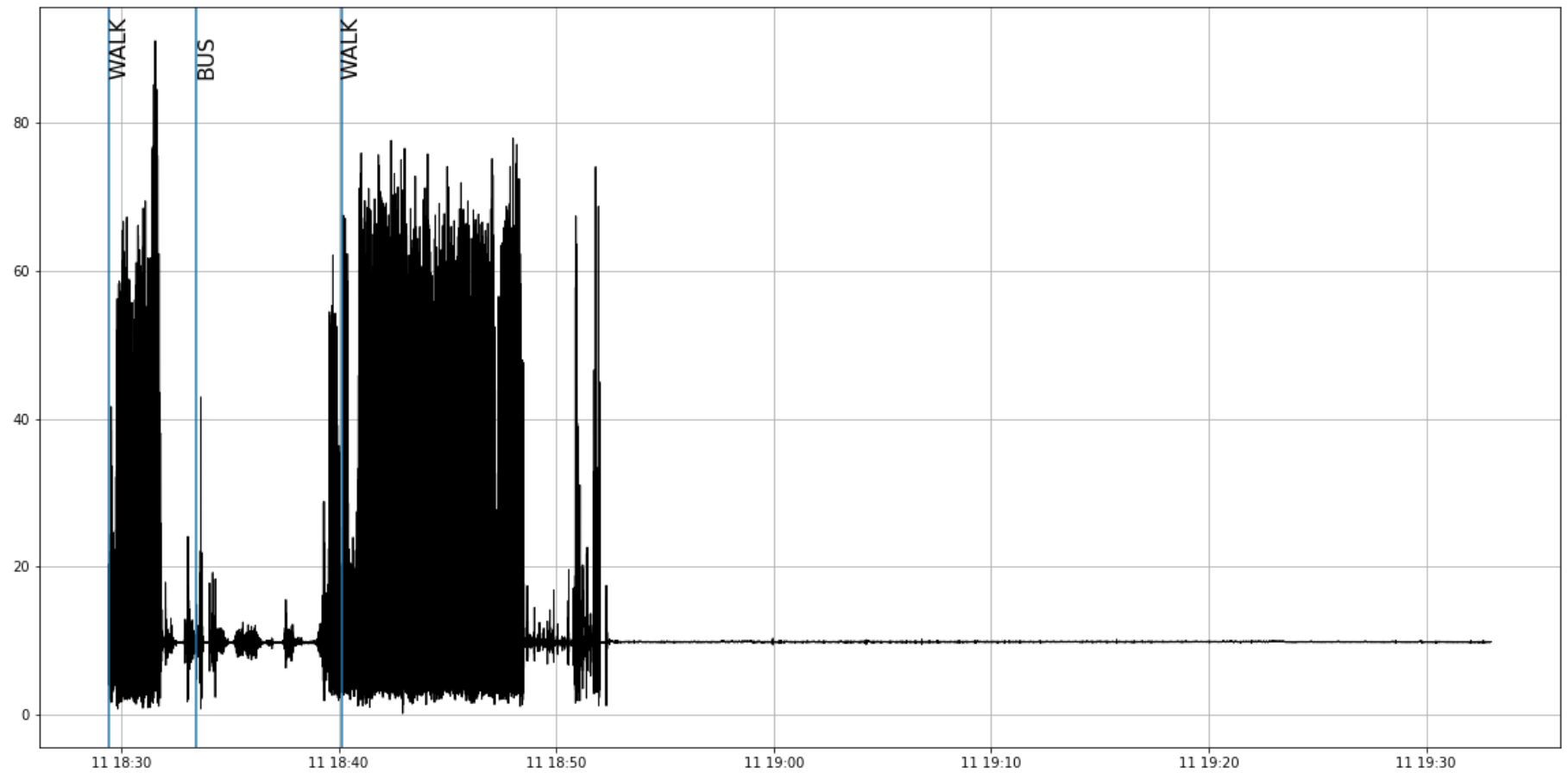




Trip id 117

Duration: 1:03:34.140000

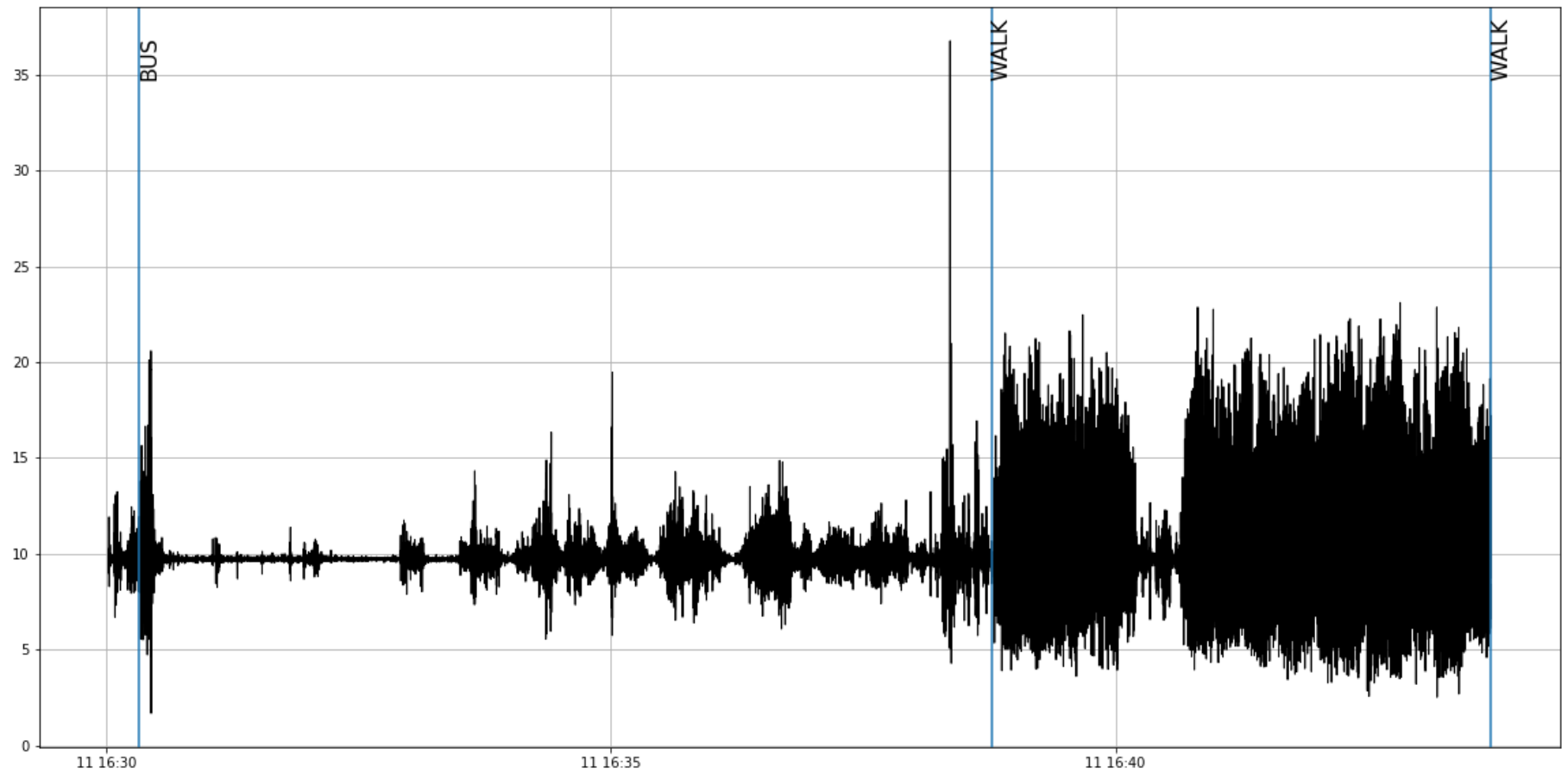
	time	key	value	mode	longitude	latitude	col7	col8	col9
4	2019-11-11T18:29:27.277Z	CGT_MODE_CHANGED	2019-11-11T18:29:24Z	WALK	16.3414021	48.2039143	-	NaN	NaN
5	2019-11-11T18:33:28.361Z	CGT_MODE_CHANGED	2019-11-11T18:33:09Z	BUS	16.3370586	48.20629238	-	Thaliastraße, Koppstraße	1.328955e+09
6	2019-11-11T18:40:07.639Z	CGT_MODE_CHANGED	2019-11-11T18:40:04Z	WALK	16.3181621	48.20899804	-	Possingergasse	3.572335e+09



Trip id 113

Duration: 0:13:41.869000

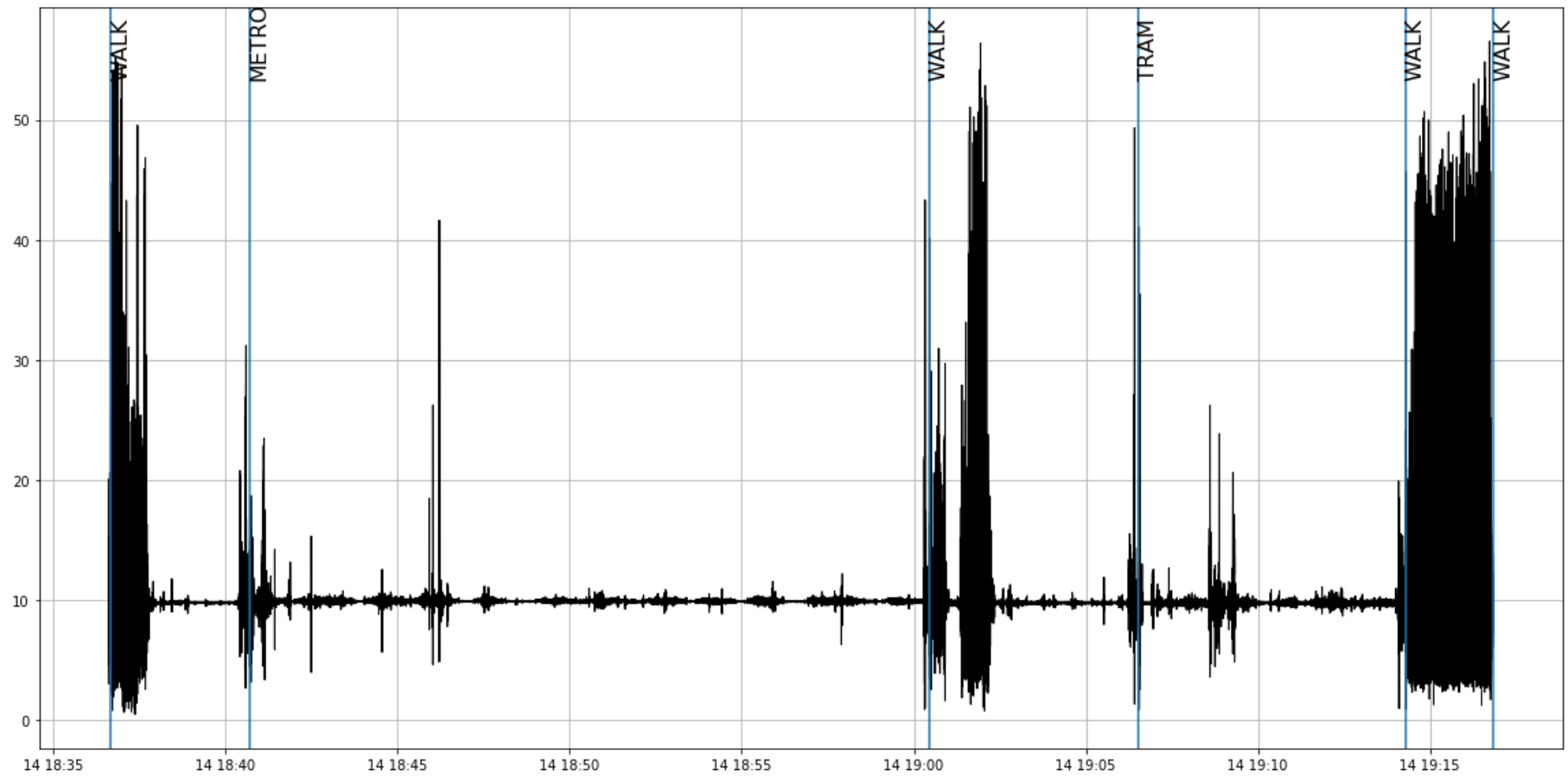
	time	key	value	mode	longitude	latitude	col7	col8	col9
4	2019-11-11T16:30:19.561Z	CGT_MODE_CHANGED	2019-11-11T16:30:01Z	BUS	16.37043654	48.21673186	-	Schottenring	559267452.0
5	2019-11-11T16:38:46.466Z	CGT_MODE_CHANGED	2019-11-11T16:38:28Z	WALK	16.37233311	48.20989157	-	Stephansplatz	7460824.0
6	2019-11-11T16:43:43.218Z	CGT_MODE_CHANGED	2019-11-11T16:43:41Z	WALK	16.37245473	48.20872625	-	NaN	NaN

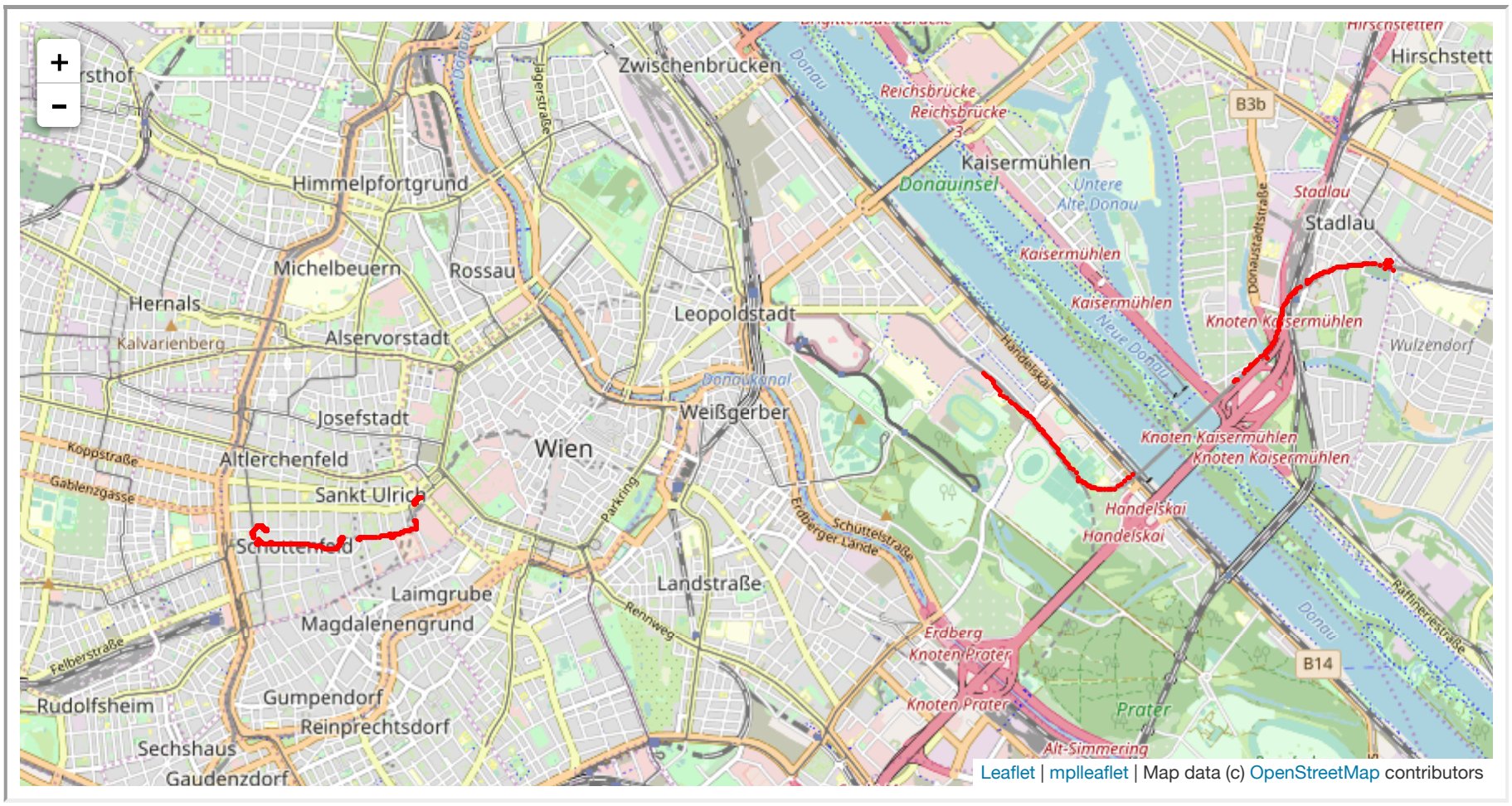


Trip id 227

Duration: 0:40:14.452000

	time	key	value	mode	longitude	latitude	col7	col8	col9
4	2019-12-14T18:36:40.328Z	CGT_MODE_CHANGED	2019-12-14T18:36:35Z	WALK	16.45760017	48.22114091	-	NaN	NaN
5	2019-12-14T18:40:41.782Z	CGT_MODE_CHANGED	2019-12-14T18:40:38Z	METRO	16.45751115	48.2211907	-	Hardeggasse	2493894258
6	2019-12-14T19:00:26.721Z	CGT_MODE_CHANGED	2019-12-14T19:00:22Z	WALK	16.3842135	48.2132408	-	Unknown station	unknown_station
7	2019-12-14T19:06:30.801Z	CGT_MODE_CHANGED	2019-12-14T19:06:26Z	TRAM	16.3570246	48.20469293	-	Volkstheater	2211384239
8	2019-12-14T19:14:16.792Z	CGT_MODE_CHANGED	2019-12-14T19:14:11Z	WALK	16.34083216	48.20189963	-	Kaiserstraße/Westbahnstraße	2293993846
9	2019-12-14T19:16:50.307Z	CGT_MODE_CHANGED	2019-12-14T19:16:49Z	WALK	16.34171046	48.20277391	-	NaN	NaN





Billie Rosalie Postlmayr

```
In [15]: postlmayr_trip_ids = {84, 93, 235, 236}
```

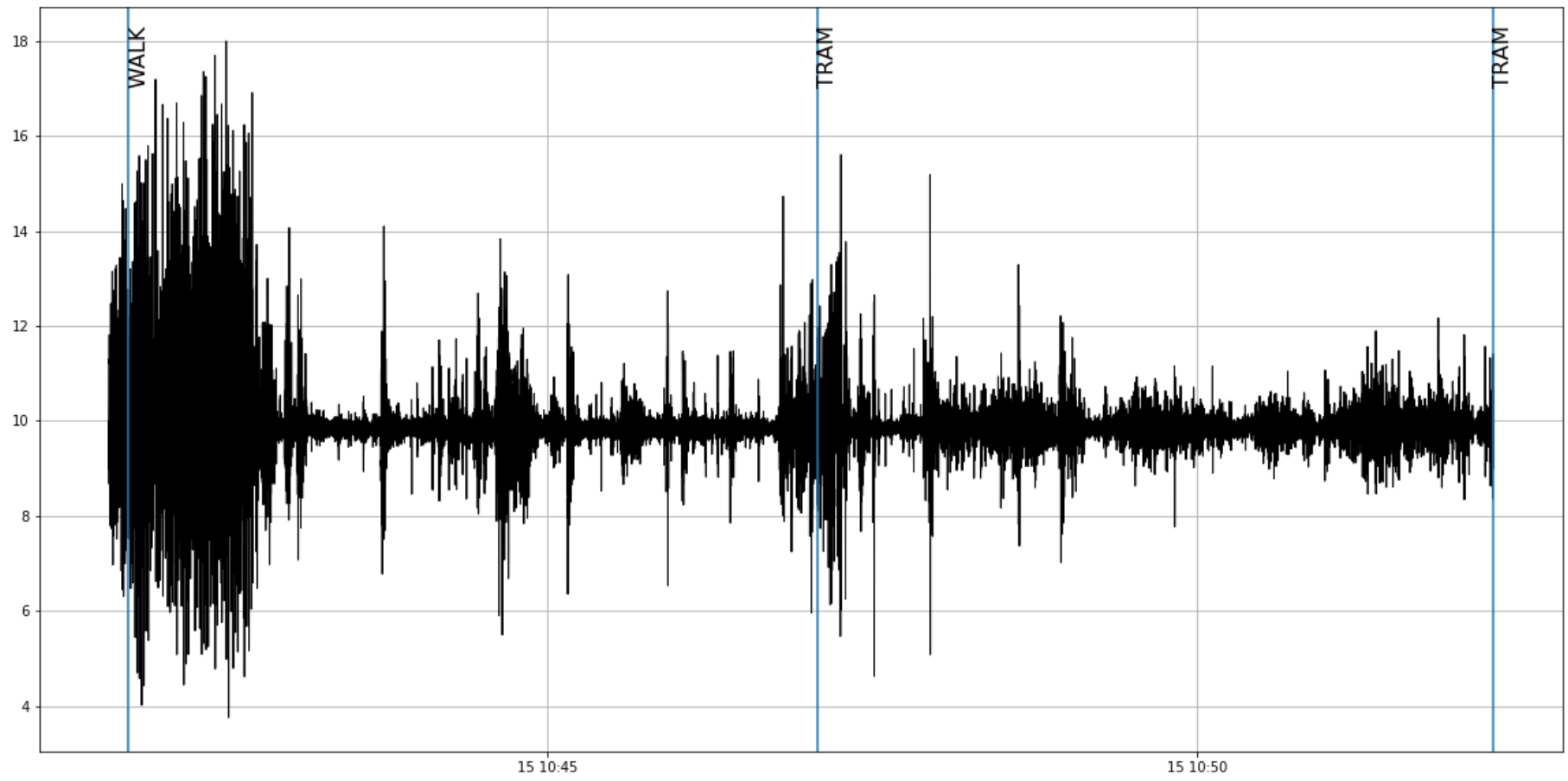
In [16]:
```
trip_overview(postlmayr_trip_ids)
```

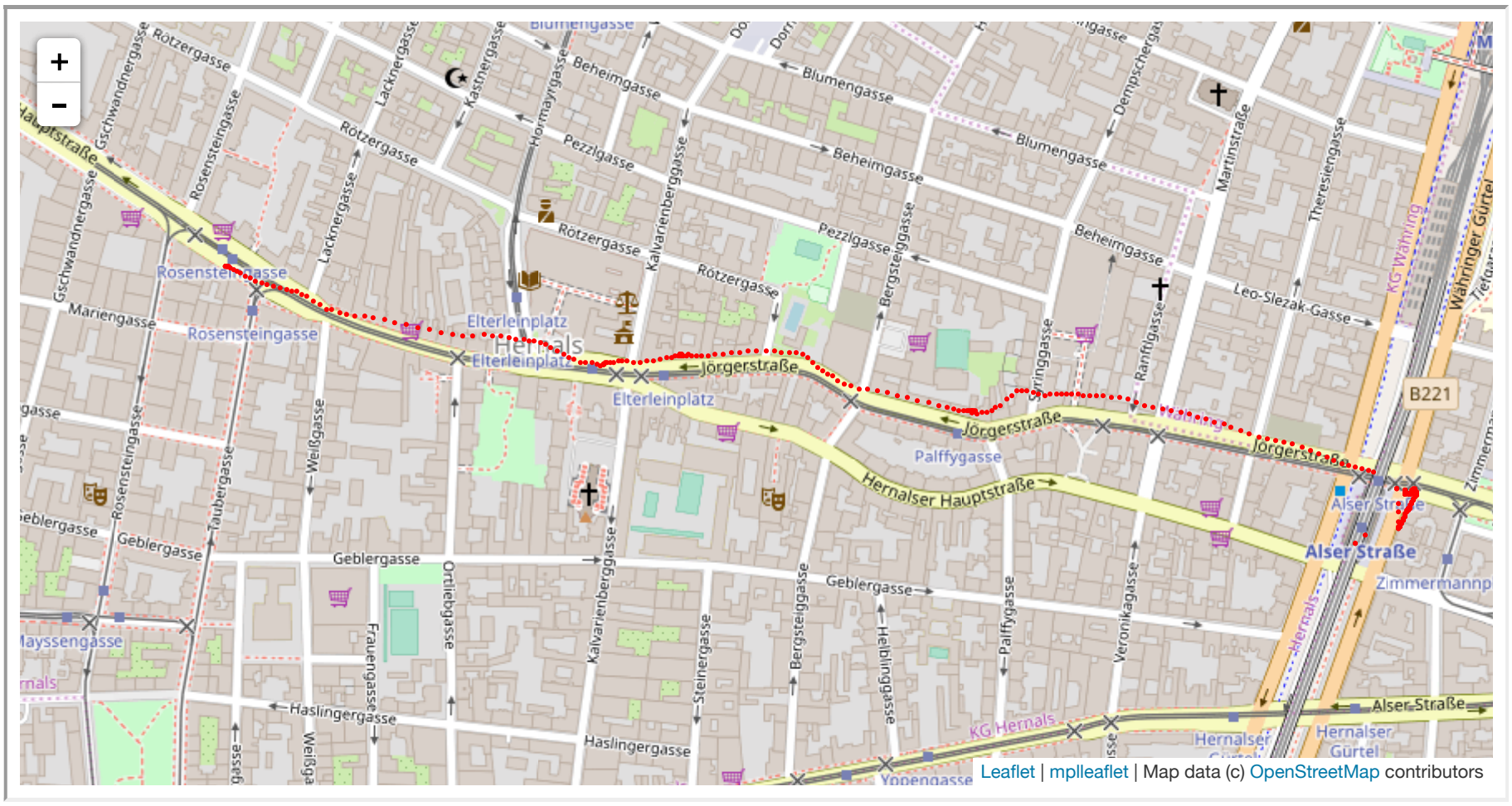
Number of trips: 4

Trip id 236

Duration: 0:10:39.424000

	time	key	value	mode	longitude	latitude	col7	col8	col9
4	2019-12-15T10:41:46.758Z	CGT_MODE_CHANGED	2019-12-15T10:41:37Z	WALK	16.3417394	48.2166381	-	NaN	NaN
5	2019-12-15T10:47:04.963Z	CGT_MODE_CHANGED	2019-12-15T10:46:56Z	TRAM	16.342368075709786	48.21710115837203	-	Alser Straße	3.630096e+09
6	2019-12-15T10:52:16.875Z	CGT_MODE_CHANGED	2019-12-15T10:52:15Z	TRAM	16.327132627745957	48.219028865563665	-	Rosensteingasse	7.465417e+06

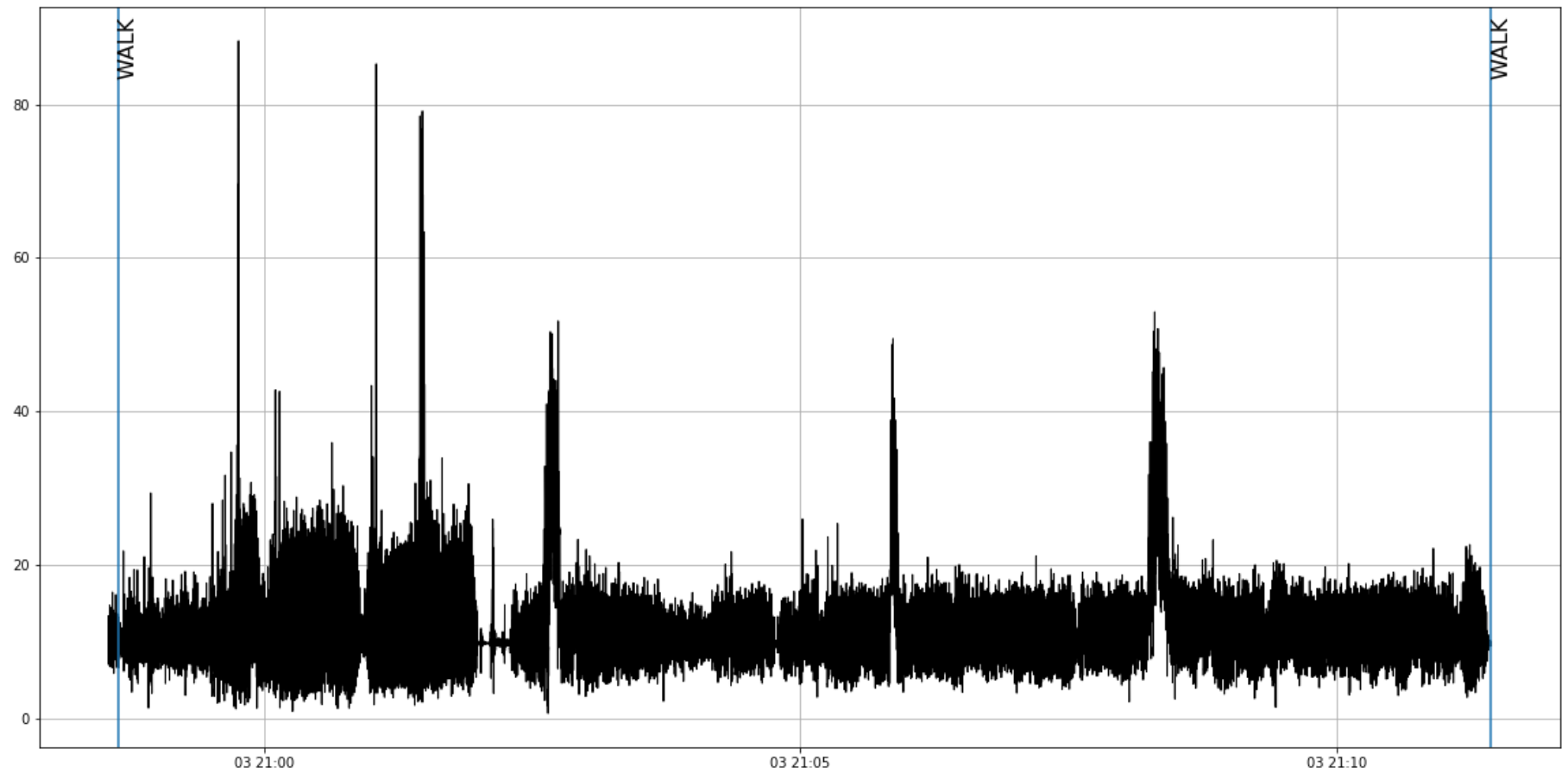


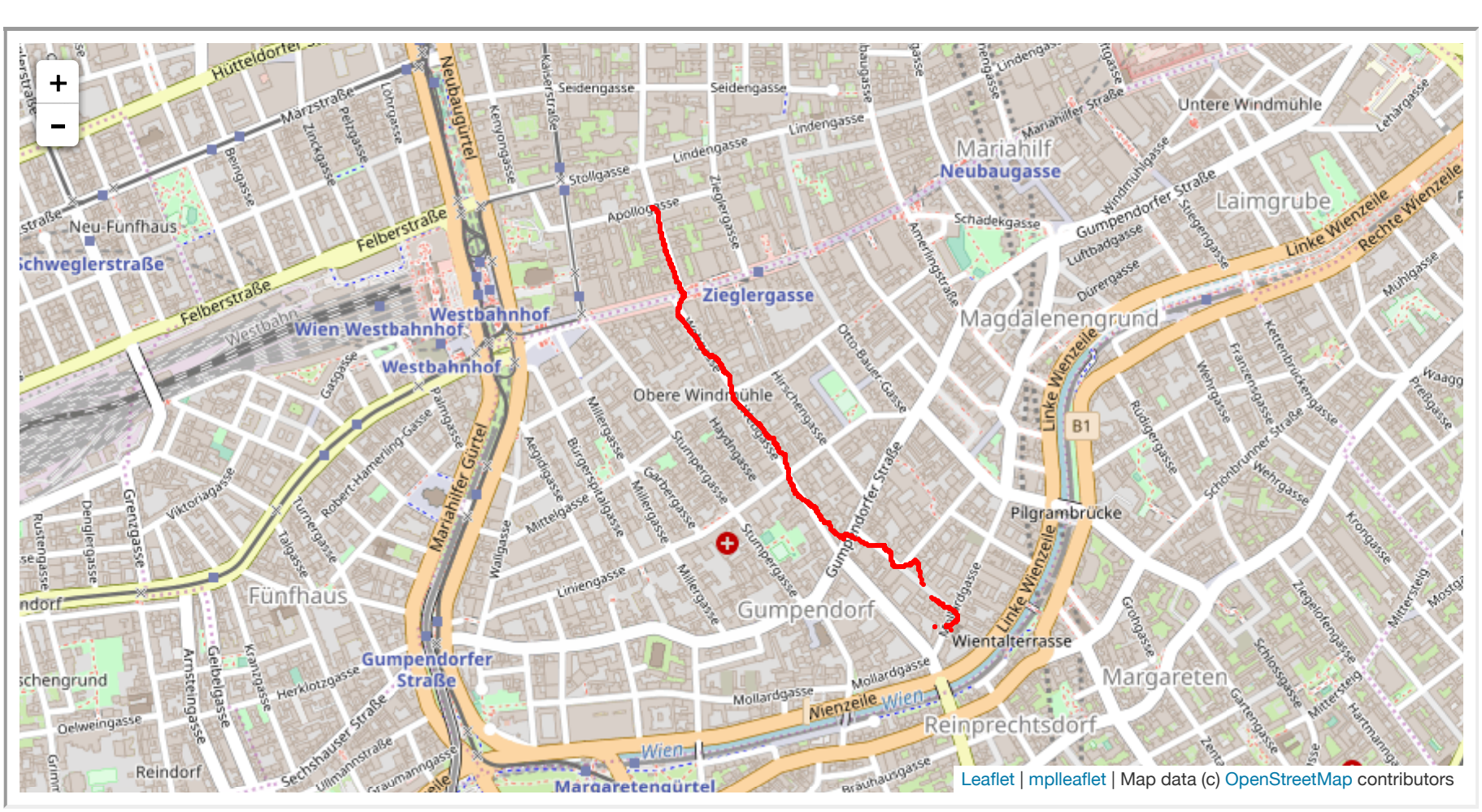


Trip id 93

Duration: 0:12:53.681000

	time	key	value	mode	longitude	latitude	col7	col8	col9
4	2019-11-03T20:58:38.457Z	CGT_MODE_CHANGED	2019-11-03T20:58:32Z	WALK	16.3509276	48.1908943	-	NaN	NaN
5	2019-11-03T21:11:26.420Z	CGT_MODE_CHANGED	2019-11-03T21:11:23Z	WALK	16.34346355548562	48.19827902411811	-	NaN	NaN

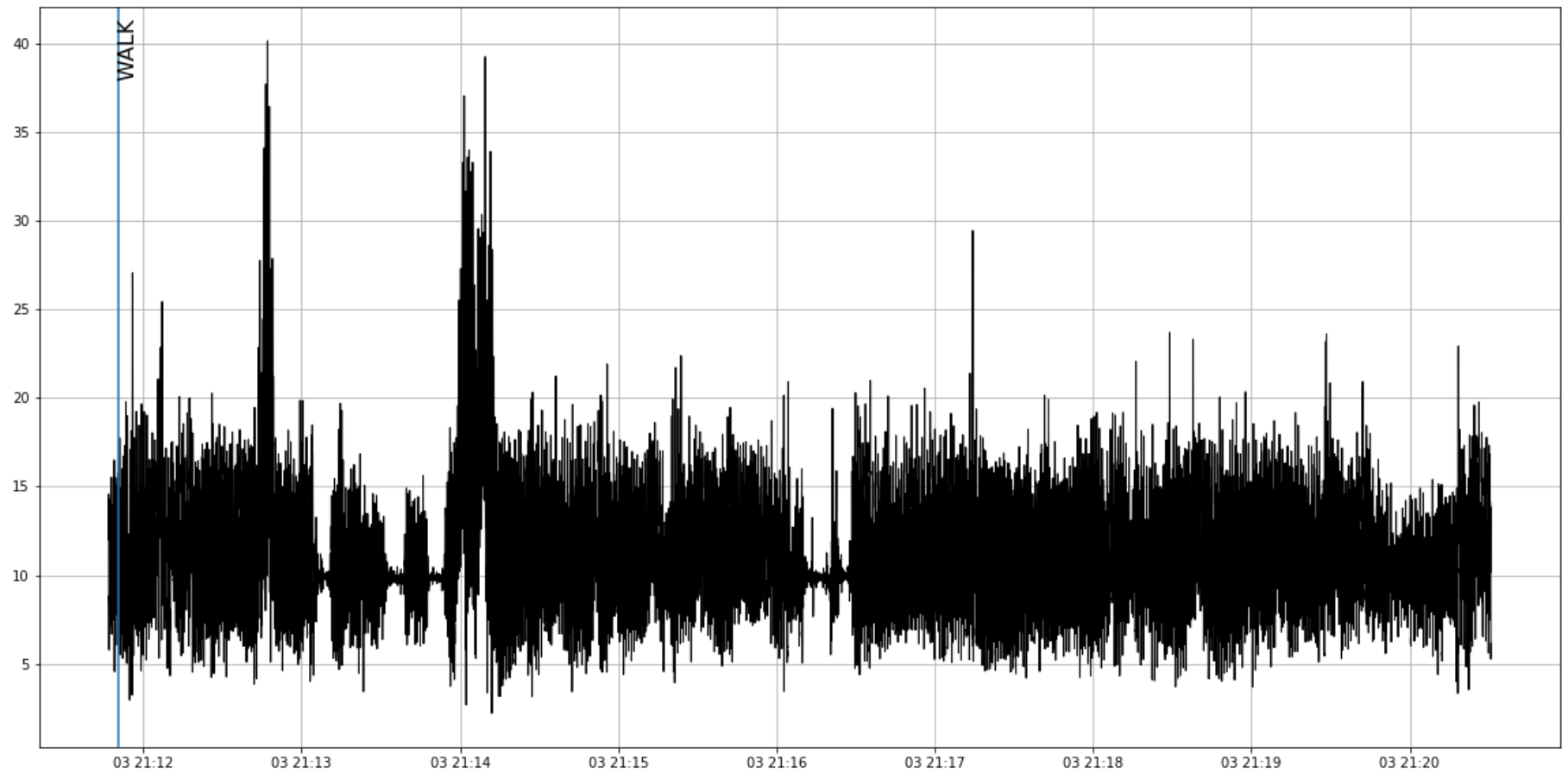


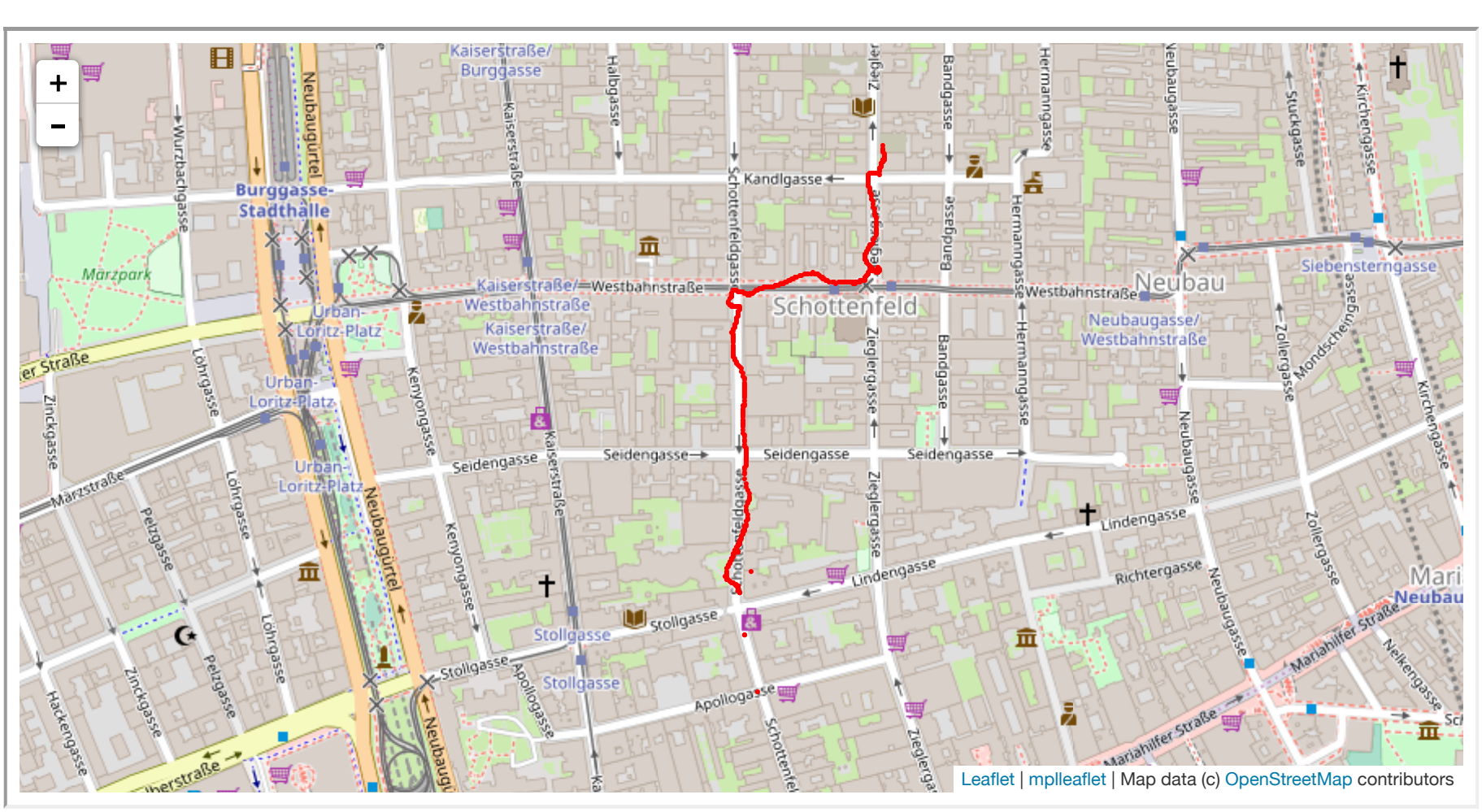


Trip id 84

Duration: 0:08:44.059000

	time	key	value	mode	longitude	latitude	col7	col8	col9
4	2019-11-03T21:11:50.515Z	CGT_MODE_CHANGED	2019-11-03T21:11:46Z	WALK	16.34346355548562	48.19827902411811	-	NaN	NaN

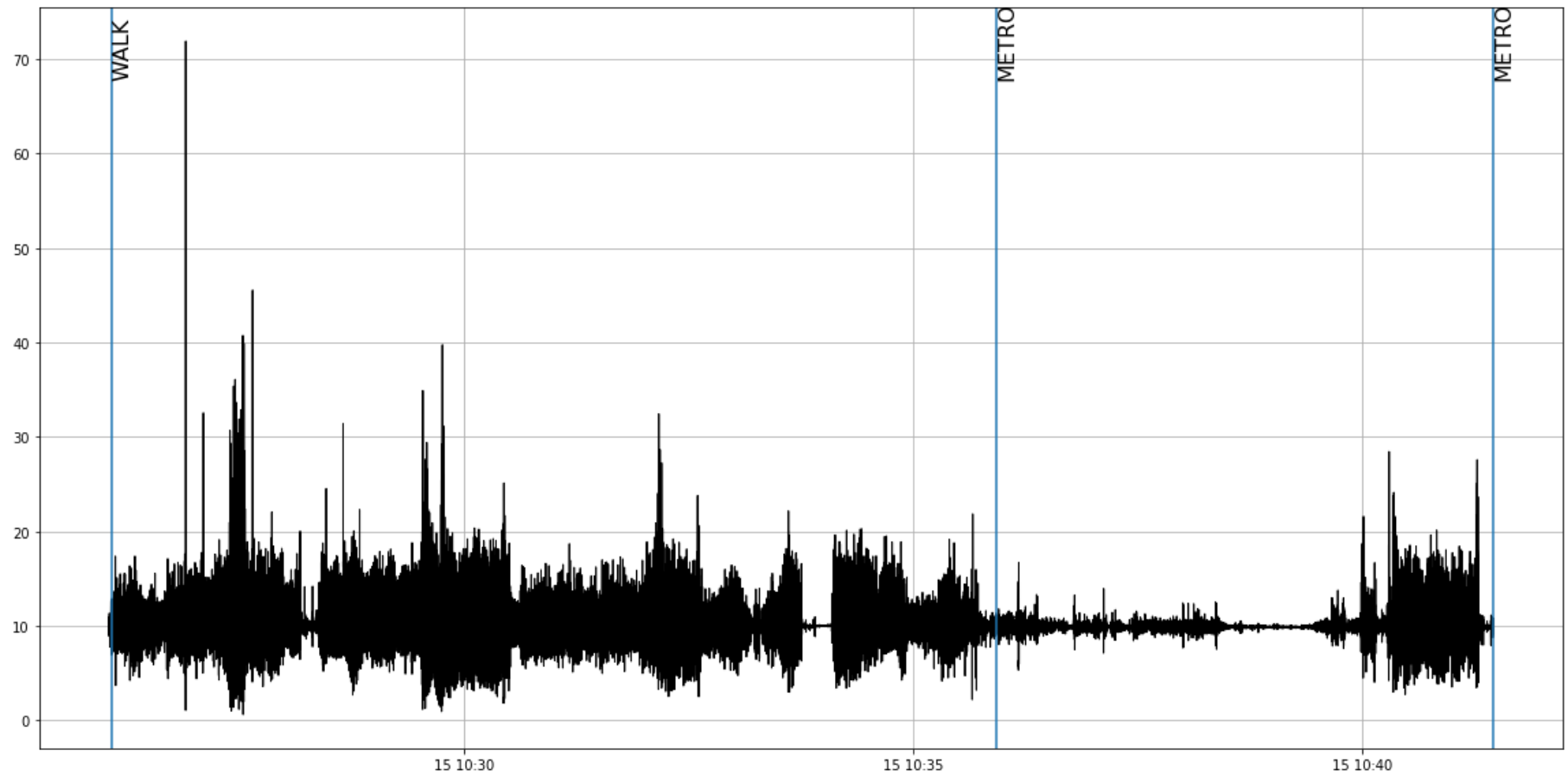


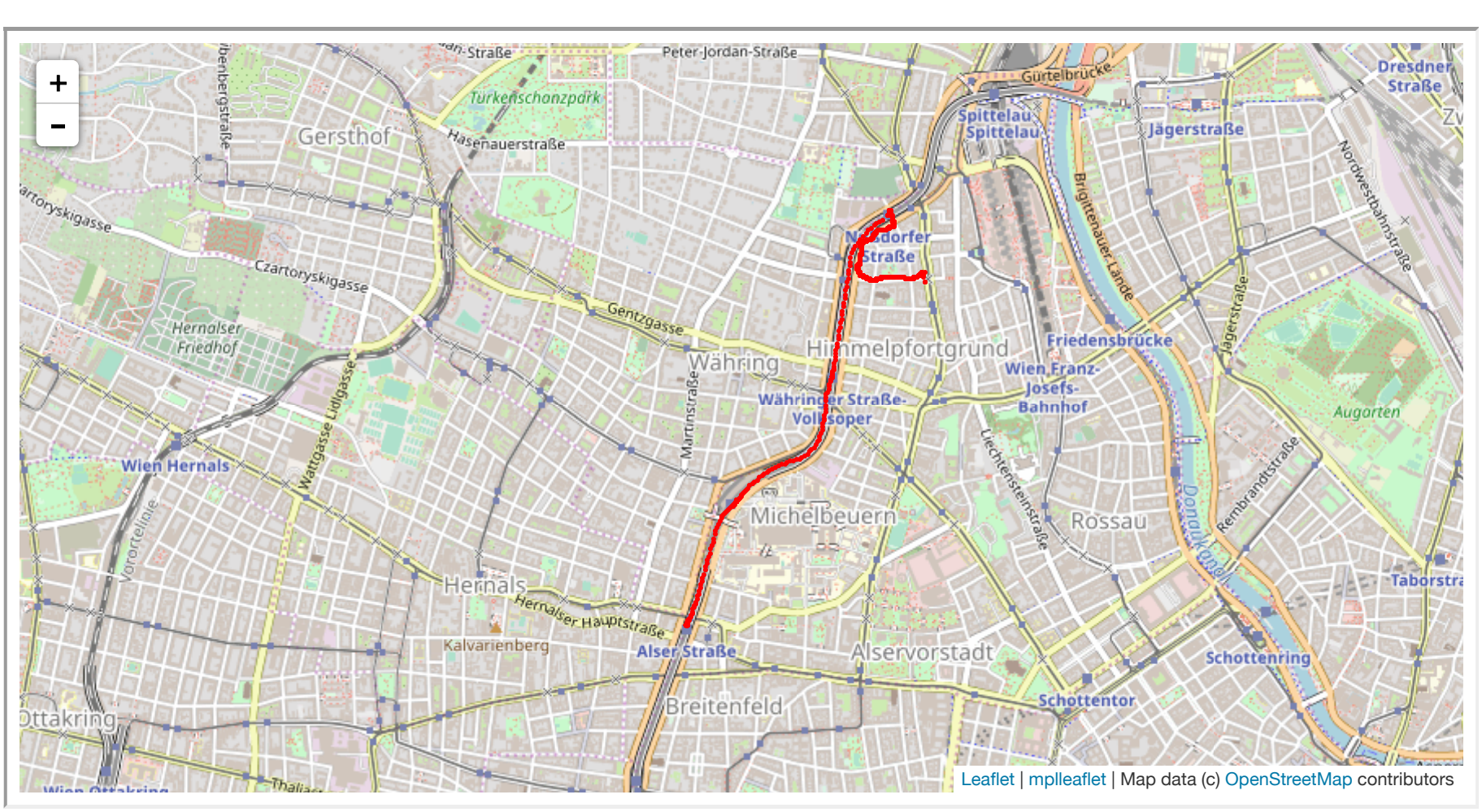


Trip id 235

Duration: 0:15:25.380000

	time	key	value	mode	longitude	latitude	col7	col8	col9
4	2019-12-15T10:26:04.551Z	CGT_MODE_CHANGED	2019-12-15T10:26:02Z	WALK	16.354338733600425	48.22936681550694	-	NaN	NaN
5	2019-12-15T10:35:55.351Z	CGT_MODE_CHANGED	2019-12-15T10:35:48Z	METRO	16.351674403419512	48.23099475319664	-	Nußdorfer Straße	3.237286e+09
6	2019-12-15T10:41:27.467Z	CGT_MODE_CHANGED	2019-12-15T10:41:26Z	METRO	16.341755389362334	48.21697815109228	-	Wien Alser Straße	6.386601e+06





2. Data Preprocessing

The following script loops over all whitelisted trips and performs the following preprocessing steps:

- Calculating the norm of the x, y and z dimensions of the acceleration data
- Downsampling the acceleration data
- Removing bimodal segments
- Combining all segments into one file (`export.csv`)

`export.csv` is then in turn used by the next script to train the model.

In [17]:
```
def replace_date(s):
    return s.group(0).replace('Z', '') + '.000Z'
```

In [18]:
```
def format_time(m):
    m = [re.sub(r':\d\d+Z', replace_date, sample) for sample in m]
    m = [float(datetime.strptime(sample, "%Y-%m-%dT%H:%M:%S.%fZ").strftime('%s.%f')) for sample in m]
    return m
```

In [19]:
```
processed = None

processed_count = 0
total_trips = len(whitelisted_trips)

for trip in whitelisted_trips:
    user_id = re.search('\d+', trip).group(0)
    trip_id = re.search('_\d+_', trip).group(0)
    trip_id = re.search('\d+', trip_id).group(0)

    processed_count = processed_count + 1
    print("processing user: ", user_id, ", trip: ", trip_id, " --- (", processed_count, "/", total_trips, ")"
)

    # markers
    path_markers = os.path.join(data_dir, trip, 'markers.csv')
    col_names = ["value", "key", "time", "mode", "col5", "col6", "col7", "station", "col9"]
    markers = pd.read_csv(path_markers, sep=';', names=col_names, skiprows=4)
    markers = markers.drop(['value', 'key', 'col5', 'col6', 'col7', 'col9'], axis=1)
    markers.drop(markers.tail(1).index, inplace=True)

    # acceleration
    path_acc = os.path.join(data_dir, trip, 'acceleration.csv')
    acceleration = pd.read_csv(path_acc, sep=',')
    acceleration['acc_norm'] = np.linalg.norm(acceleration[['x', 'y', 'z']].values, axis=1)
    acceleration = acceleration.drop(['x', 'y', 'z'], axis=1)

    # activity
#     path_activity = os.path.join(data_dir, trip, 'activity_records.csv')
#     activity = pd.read_csv(path_activity, sep=',')

    # time formatting
    acceleration['time'] = format_time(acceleration['time'])
    acceleration['time'] = acceleration['time'].astype(int)
    markers['time'] = format_time(markers['time'])
    markers['time'] = markers['time'].astype(int)

    # downsample
    acceleration = acceleration.set_index(['time'])
    acceleration.index = pd.to_datetime(acceleration.index, unit='ms')
    acceleration = acceleration.resample('1L').mean()
    markers = markers.set_index(['time'])
    markers.index = pd.to_datetime(markers.index, unit='ms')

    # combine acceleration and markers
    df_trip = acceleration.merge(markers, on="time", how='left')
    df_trip = df_trip.ffill()

    # eliminate bimodal segments
    df_trip = df_trip.reset_index()
    drop_multimodal = []
    for i in range(int(df_trip.shape[0] / 10)):
        counts = df_trip.iloc[i * 10:i * 10 + 10].groupby('mode').count()
        if counts.shape[0] != 1:
            for val in range(i * 10, i * 10 + 10):
                drop_multimodal.append(val)
    df_trip.drop(drop_multimodal, inplace=True)

    # add ids and add to overall dataframe
    df_trip['user_id'] = user_id
    df_trip['trip_id'] = trip_id
    if processed is None:
        processed = df_trip.copy()
    else:
        processed = processed.append(df_trip, ignore_index=True, sort=False)

processed = processed.set_index(['time'])
processed.to_csv('export.csv', header=True)
```

```
processing user:  11 , trip:  248  --- ( 1 / 94 )
processing user:  30 , trip:  22  --- ( 2 / 94 )
processing user:  18 , trip:  55  --- ( 3 / 94 )
processing user:  22 , trip:  52  --- ( 4 / 94 )
processing user:  35 , trip:  240  --- ( 5 / 94 )
processing user:  18 , trip:  217  --- ( 6 / 94 )
processing user:  30 , trip:  103  --- ( 7 / 94 )
processing user:  26 , trip:  46  --- ( 8 / 94 )
processing user:  18 , trip:  9  --- ( 9 / 94 )
processing user:  27 , trip:  108  --- ( 10 / 94 )
processing user:  31 , trip:  38  --- ( 11 / 94 )
processing user:  10 , trip:  160  --- ( 12 / 94 )
processing user:  32 , trip:  236  --- ( 13 / 94 )
processing user:  30 , trip:  42  --- ( 14 / 94 )
processing user:  14 , trip:  233  --- ( 15 / 94 )
processing user:  10 , trip:  208  --- ( 16 / 94 )
processing user:  11 , trip:  246  --- ( 17 / 94 )
processing user:  11 , trip:  241  --- ( 18 / 94 )
processing user:  14 , trip:  110  --- ( 19 / 94 )
processing user:  30 , trip:  215  --- ( 20 / 94 )
processing user:  35 , trip:  115  --- ( 21 / 94 )
processing user:  35 , trip:  116  --- ( 22 / 94 )
processing user:  32 , trip:  93  --- ( 23 / 94 )
processing user:  13 , trip:  220  --- ( 24 / 94 )
processing user:  28 , trip:  152  --- ( 25 / 94 )
processing user:  18 , trip:  88  --- ( 26 / 94 )
processing user:  5 , trip:  5  --- ( 27 / 94 )
processing user:  10 , trip:  82  --- ( 28 / 94 )
processing user:  26 , trip:  74  --- ( 29 / 94 )
processing user:  26 , trip:  129  --- ( 30 / 94 )
processing user:  26 , trip:  127  --- ( 31 / 94 )
processing user:  24 , trip:  40  --- ( 32 / 94 )
processing user:  35 , trip:  114  --- ( 33 / 94 )
processing user:  30 , trip:  102  --- ( 34 / 94 )
processing user:  24 , trip:  36  --- ( 35 / 94 )
processing user:  19 , trip:  139  --- ( 36 / 94 )
processing user:  28 , trip:  43  --- ( 37 / 94 )
processing user:  10 , trip:  83  --- ( 38 / 94 )
processing user:  11 , trip:  245  --- ( 39 / 94 )
processing user:  13 , trip:  223  --- ( 40 / 94 )
processing user:  28 , trip:  28  --- ( 41 / 94 )
processing user:  19 , trip:  19  --- ( 42 / 94 )
processing user:  6 , trip:  47  --- ( 43 / 94 )
processing user:  18 , trip:  211  --- ( 44 / 94 )
processing user:  35 , trip:  226  --- ( 45 / 94 )
processing user:  19 , trip:  142  --- ( 46 / 94 )
processing user:  27 , trip:  109  --- ( 47 / 94 )
processing user:  14 , trip:  234  --- ( 48 / 94 )
processing user:  19 , trip:  134  --- ( 49 / 94 )
processing user:  18 , trip:  35  --- ( 50 / 94 )
processing user:  19 , trip:  137  --- ( 51 / 94 )
processing user:  35 , trip:  219  --- ( 52 / 94 )
processing user:  13 , trip:  224  --- ( 53 / 94 )
processing user:  31 , trip:  70  --- ( 54 / 94 )
processing user:  6 , trip:  72  --- ( 55 / 94 )
processing user:  26 , trip:  131  --- ( 56 / 94 )
processing user:  18 , trip:  210  --- ( 57 / 94 )
processing user:  35 , trip:  218  --- ( 58 / 94 )
processing user:  14 , trip:  237  --- ( 59 / 94 )
processing user:  31 , trip:  32  --- ( 60 / 94 )
processing user:  24 , trip:  49  --- ( 61 / 94 )
processing user:  24 , trip:  33  --- ( 62 / 94 )
processing user:  24 , trip:  37  --- ( 63 / 94 )
processing user:  27 , trip:  99  --- ( 64 / 94 )
processing user:  28 , trip:  203  --- ( 65 / 94 )
processing user:  31 , trip:  199  --- ( 66 / 94 )
processing user:  35 , trip:  117  --- ( 67 / 94 )
processing user:  30 , trip:  147  --- ( 68 / 94 )
processing user:  30 , trip:  24  --- ( 69 / 94 )
processing user:  35 , trip:  113  --- ( 70 / 94 )
processing user:  32 , trip:  84  --- ( 71 / 94 )
processing user:  30 , trip:  214  --- ( 72 / 94 )
processing user:  22 , trip:  145  --- ( 73 / 94 )
processing user:  26 , trip:  165  --- ( 74 / 94 )
processing user:  27 , trip:  107  --- ( 75 / 94 )
processing user:  14 , trip:  228  --- ( 76 / 94 )
processing user:  27 , trip:  105  --- ( 77 / 94 )
processing user:  6 , trip:  204  --- ( 78 / 94 )
processing user:  10 , trip:  161  --- ( 79 / 94 )
processing user:  24 , trip:  120  --- ( 80 / 94 )
processing user:  26 , trip:  128  --- ( 81 / 94 )
processing user:  28 , trip:  13  --- ( 82 / 94 )
processing user:  10 , trip:  78  --- ( 83 / 94 )
processing user:  35 , trip:  227  --- ( 84 / 94 )
processing user:  31 , trip:  95  --- ( 85 / 94 )
processing user:  10 , trip:  209  --- ( 86 / 94 )
processing user:  28 , trip:  202  --- ( 87 / 94 )
processing user:  30 , trip:  26  --- ( 88 / 94 )
```

```
processing user:  32 , trip:  235  --- ( 89 / 94 )
processing user:  8 , trip:  91  --- ( 90 / 94 )
processing user:  31 , trip:  21  --- ( 91 / 94 )
processing user:  22 , trip:  146  --- ( 92 / 94 )
processing user:  6 , trip:  201  --- ( 93 / 94 )
processing user:  13 , trip:  222  --- ( 94 / 94 )
```

3. Insights

The data set consists of 94 trips by 17 students.

In [20]:
```
export = pd.read_csv(os.path.join("export.csv"))
```

The preprocessed data consists of the following entry counts per transportation mode:

In [21]:
```
counts = export.groupby('mode').agg(['count']).stack()['time']
print(counts)
```

```
mode
2         count      430
BICYCLE   count     2423
BUS       count    12197
CAR       count    35054
METRO     count    20020
TRAIN     count     1739
TRAM      count    23223
WALK      count    55880
Name: time, dtype: int64
```

In [22]:
```
percent = counts/export.count()['time']
percent
```

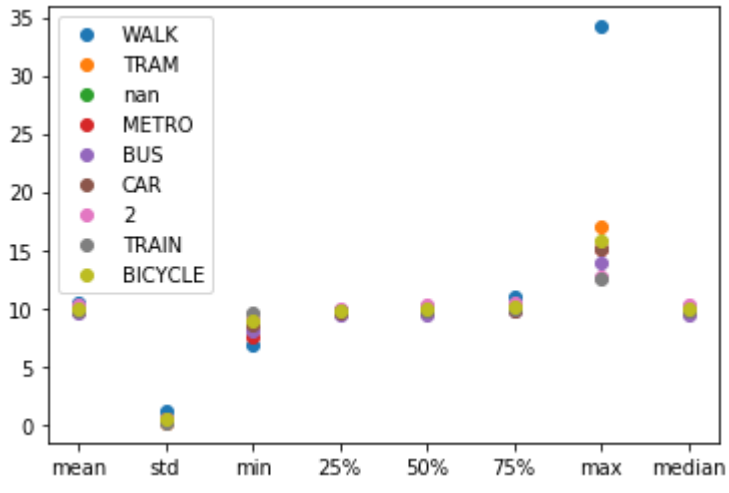
Out[22]:
```
mode
2         count    0.002846
BICYCLE   count    0.016035
BUS       count    0.080720
CAR       count    0.231987
METRO     count    0.132492
TRAIN     count    0.011509
TRAM      count    0.153690
WALK      count    0.369814
Name: time, dtype: float64
```

It looks like the different transportation modes are nearly indistinguishable when only considering the mean, std and quartiles:

In [23]:
```
for mode in export['mode'].unique():
    print("Mode:", mode)
    mode_norm = export[export['mode'] == mode]['acc_norm']
    desc = mode_norm.describe().drop('count')
    desc['median'] = mode_norm.median()
    print(desc)
    plt.plot(desc, label=mode, marker='o', linestyle='none')
    print()

plt.legend()
plt.show()
```

```
Mode: WALK
mean      10.519006
std        1.110724
min        6.781093
25%        9.825099
50%       10.204862
75%       10.962832
max       34.183788
median    10.204862
Name: acc_norm, dtype: float64

Mode: TRAM
mean       9.822176
std        0.232906
min        8.383940
25%        9.750623
50%        9.828687
75%        9.878594
max       16.963387
median     9.828687
Name: acc_norm, dtype: float64

Mode: nan
mean     NaN
std      NaN
min      NaN
25%      NaN
50%      NaN
75%      NaN
max      NaN
median   NaN
Name: acc_norm, dtype: float64

Mode: METRO
mean       9.732295
std        0.426413
min        7.624484
25%        9.732897
50%        9.824142
75%        9.906877
max       15.173608
median     9.824142
Name: acc_norm, dtype: float64

Mode: BUS
mean       9.549863
std        0.363815
min        7.990282
25%        9.379823
50%        9.495853
75%        9.733191
max       13.923279
median     9.495853
Name: acc_norm, dtype: float64

Mode: CAR
mean       9.711362
std        0.247312
min        8.570905
25%        9.548012
50%        9.720075
75%        9.822520
max       15.183380
median     9.720075
Name: acc_norm, dtype: float64

Mode: 2
mean      10.219519
std        0.442404
min        9.091292
25%       10.003307
50%       10.251154
75%       10.440192
max       12.694286
median    10.251154
Name: acc_norm, dtype: float64

Mode: TRAIN
mean       9.824039
std        0.163547
min        9.558453
25%        9.704136
50%        9.850618
75%        9.904287
max       12.520542
median     9.850618
Name: acc_norm, dtype: float64
```

```
Mode: BICYCLE
mean      10.001763
std        0.415933
min        8.916595
25%        9.767371
50%        9.925378
75%       10.132358
max       15.731389
median     9.925378
Name: acc_norm, dtype: float64
```



4. Classification

After a bit of research[1][2], we concluded that a Convolutional Neural Network (CNN) whould be best suited for time series-based Human Activity Recognition (HAR) tasks, as we have here.

Optimizers[3]

We tried using different optimizers for our sequential Keras model, with varying results. The stochastic gradient descent (SGD) optimizer resulted in an accuracy below .8 and the `adam` optimizer managed to reach an accuracy of .82 with a variance of .05. But ultimately, the best performance was achieved using the `Nadam` [4] optimizer with default parameters. Nadam is short for "Nesterov Adam optimizer", which is essentially RMSprop with Nesterov momentum. This optimizer resulted in an accuracy of about .884 with little variance, due to the fact that this optimizer is uniquely suited to our use case.

Loss Function[5]

As a loss function for our model, we used binary crossentropy since it proved to be more accurate than categorical crossenropy.

Sample size of time window

A sample size of `n = 10` resulted in an accuracy of ~.88, and `n = 100` resulted in a very stable accuracy of ~.92. The best accuracy was achieved with a window of `n = 20` seconds, which resulted in ~.97 accuracy with a variance of ~0.01 between the results.

In [27]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.utils import to_categorical
from scipy import fft
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_fscore_support
```

`export_mobility.csv` is basically the same as `export.csv` , with the only difference that it contains the x,y,z dimensions instead of the norm of those dimensions.

In [28]:
```
data = pd.read_csv('export_mobility.csv', sep=',')
data2 = pd.read_csv('export_mobility_v2.csv', sep=',')
data = data[['x', 'y', 'z', 'mode']]
data2 = data2[['x', 'y', 'z', 'mode']]
data = data.append(data2)
data = data.reset_index()
modes = list(data.groupby('mode').count().index[1:])
```

4.1 Single Classifier model (SCM)

4.1.1 Feature Preparation

In [29]:
```
i = 0
dict = {}
X = None
y = None
```

In [30]:
```
for mode in modes:
    if mode in ['BICYCLE', 'BUS', 'CAR', 'METRO', 'TRAIN', 'TRAM', 'WALK']:
        i = i+1
        df_mode = data[data['mode'] == mode][['x', 'y', 'z']]
        n = df_mode.shape[0]
        sample_n = 20
        rest = n % sample_n
        df_mode.drop(df_mode.tail(rest).index, inplace=True)
        n = n-rest
        Xx = np.array(df_mode['x']).reshape(int(n / sample_n), sample_n)
        xfft = fft(np.array(df_mode['x']).reshape(int(n / sample_n), sample_n), axis=0)
        Xy = np.array(df_mode['y']).reshape(int(n / sample_n), sample_n)
        yfft = fft(np.array(df_mode['y']).reshape(int(n / sample_n), sample_n), axis=0)
        Xz = np.array(df_mode['z']).reshape(int(n / sample_n), sample_n)
        zfft = fft(np.array(df_mode['z']).reshape(int(n / sample_n), sample_n), axis=0)
        XXX = np.dstack((Xx, Xy, Xz, xfft, yfft, zfft))
        if X is None:
            X = XXX
        else:
            X = np.append(X, XXX, axis=0)
        if y is None:
            y = np.full((int(n/sample_n)), i)
        else:
            y = np.append(y, np.full((int(n/sample_n)), i))
        dict[mode] = i
y = to_categorical(y)
```

4.1.2 Creating and training the model

In [31]:
```
model_best = None
score_best = None

kfold = KFold(5, True, 1)

for train, test in kfold.split(X):

    x_train, x_test, y_train, y_test = X[train], X[test], y[train], y[test]

    verbose, epochs, batch_size = 0, 10, 32
    samples, features, outputs = x_train.shape[1], x_train.shape[2], y_train.shape[1]

    model = Sequential()

    model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(samples, features)))
    model.add(Conv1D(filters=64, kernel_size=2, activation='relu'))
    model.add(Dropout(0.5))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(outputs, activation='softmax'))
    model.compile(loss='binary_crossentropy', optimizer='nadam', metrics=['accuracy'])

    model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, verbose=verbose)

    # evaluate the model
    _, accuracy_train = model.evaluate(x_train, y_train, batch_size=batch_size, verbose=0)
    _, accuracy_test = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=0)

    if score_best is None:
        score_best = accuracy_test
    elif accuracy_test > score_best:
        score_best = accuracy_test
        model_best = model

    print("training accuracy: ", accuracy_train, "testing accuracy: ", accuracy_test)

#    print(precision_recall_fscore_support(y_true, y_pred))
```

```
/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

training accuracy:  0.9905783534049988 testing accuracy:  0.9735074639320374

/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

training accuracy:  0.9899253845214844 testing accuracy:  0.9783582091331482

/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

training accuracy:  0.9908581972122192 testing accuracy:  0.9738805890083313

/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

training accuracy:  0.9864738583564758 testing accuracy:  0.9649253487586975

/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

training accuracy:  0.9947761297225952 testing accuracy:  0.9750000238418579
```

4.2 Ensemble Walk Classifier model (EWCM)

The EWCM is a staged approach where first a model is trained to decide whether a trip segment is walk or non-walk, and then a second model is trained to classify all the non-walk segments.

4.2.1 Feature Preparation

In [32]:
```
i = 0
dict = {}

X_walk = None
X_nowalk = None
y_nowalk = None
y_walk = None
y = None

sum = 0
```

In [33]:
```
for mode in modes:
    if mode in ['BICYCLE', 'BUS', 'CAR', 'METRO', 'TRAM', 'WALK']:
        df_mode = data[data['mode'] == mode][['x', 'y', 'z']]
        n = df_mode.shape[0]
        sample_n = 20
        sum = int(sum + n/sample_n)
        print(sum)
        rest = n % sample_n
        df_mode.drop(df_mode.tail(rest).index, inplace=True)
        n = n-rest
        Xx = np.array(df_mode['x']).reshape(int(n / sample_n), sample_n)
        xfft = fft(np.array(df_mode['x']).reshape(int(n / sample_n), sample_n), axis=0)
        Xy = np.array(df_mode['y']).reshape(int(n / sample_n), sample_n)
        yfft = fft(np.array(df_mode['y']).reshape(int(n / sample_n), sample_n), axis=0)
        Xz = np.array(df_mode['z']).reshape(int(n / sample_n), sample_n)
        zfft = fft(np.array(df_mode['z']).reshape(int(n / sample_n), sample_n), axis=0)
        XXX = np.dstack((Xx, Xy, Xz, xfft, yfft, zfft))

        # assign X for both cases
        if X_walk is None:
            X_walk = XXX
        if X_nowalk is None and mode != "WALK":
            X_nowalk = XXX
        else:
            X_walk = np.append(X_walk, XXX, axis=0)
            if mode != "WALK":
                X_nowalk = np.append(X_nowalk, XXX, axis=0)

        # assign y for both cases as well as the objective one
        if y is None:
            y = np.full((int(n / sample_n)), i)
        else:
            y = np.append(y, np.full((int(n/sample_n)), i))

        if y_walk is None:
            if mode == "WALK":
                y_walk = np.full((int(n / sample_n)), 1)
            else:
                y_walk = np.full((int(n / sample_n)), 0)
        if y_nowalk is None and mode != "WALK":
            y_nowalk = np.full((int(n/sample_n)), i)
        else:
            if mode == "WALK":
                y_walk = np.append(y_walk,np.full((int(n/sample_n)), 1))
            else:
                y_walk = np.append(y_walk, np.full((int(n/sample_n)), 0))
                y_nowalk = np.append(y_nowalk, np.full((int(n/sample_n)), i))
        dict[mode] = i
        i = i+1

y_nowalk = to_categorical(y_nowalk)
y_walk = to_categorical(y_walk)
```

```
28
78
147
386
628
1663
```

4.2.2 Creating and training the walk model

In [34]:
```
model_walk = None
score_walk = None

kfold = KFold(5, True, 1)

for train, test in kfold.split(X_walk):

    x_train, x_test, y_train, y_test = X_walk[train], X_walk[test], y_walk[train], y_walk[test]

    verbose, epochs, batch_size = 0, 10, 32
    samples, features, outputs = x_train.shape[1], x_train.shape[2], y_train.shape[1]

    model = Sequential()

    model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(samples, features)))
    model.add(Conv1D(filters=64, kernel_size=2, activation='relu'))
    model.add(Dropout(0.5))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(outputs, activation='softmax'))

    model.compile(loss='binary_crossentropy', optimizer='nadam', metrics=['accuracy'])
    model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, verbose=verbose)

    # evaluate the model
    _, accuracy_test = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=0)
    _, accuracy_train = model.evaluate(x_train, y_train, batch_size=batch_size, verbose=0)

    if score_walk is None:
        score_walk = accuracy_test
    elif accuracy_test > score_walk:
        score_walk = accuracy_test
        model_walk = model

    print("testing accuracy: ", accuracy_test, "training accuracy: ", accuracy_train)

#    print(precision_recall_fscore_support(y_true, y_pred))
```

```
/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

testing accuracy:  0.9639639854431152 training accuracy:  0.9842105507850647

/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

testing accuracy:  0.9759759902954102 training accuracy:  0.9917293190956116

/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

testing accuracy:  0.966966986656189 training accuracy:  0.9894737005233765

/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

testing accuracy:  0.9698795080184937 training accuracy:  0.9939894676208496

/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

testing accuracy:  0.9668674468994141 training accuracy:  0.9797145128250122
```

In [35]:
```
model_walk.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_13 (Conv1D)           (None, 19, 64)            832
_________________________________________________________________
conv1d_14 (Conv1D)           (None, 18, 64)            8256
_________________________________________________________________
dropout_7 (Dropout)          (None, 18, 64)            0
_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 9, 64)             0
_________________________________________________________________
flatten_7 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_13 (Dense)             (None, 100)               57700
_________________________________________________________________
dense_14 (Dense)             (None, 2)                 202
=================================================================
Total params: 66,990
Trainable params: 66,990
Non-trainable params: 0
_________________________________________________________________
```

4.2.3 Creating and training the no-walk model

In [36]:
```
model_nowalk = None
score_nowalk = None

kfold = KFold(5, True, 1)

for train, test in kfold.split(X_nowalk):

    x_train, x_test, y_train, y_test = X_nowalk[train], X_nowalk[test], y_nowalk[train], y_nowalk[test]

    verbose, epochs, batch_size = 0, 10, 32
    samples, features, outputs = x_train.shape[1], x_train.shape[2], y_train.shape[1]

    model = Sequential()

    model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(samples, features)))
    model.add(Conv1D(filters=64, kernel_size=2, activation='relu'))
    model.add(Dropout(0.5))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(outputs, activation='softmax'))

    model.compile(loss='binary_crossentropy', optimizer='nadam', metrics=['accuracy'])
    model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size, verbose=verbose)

    # evaluate the model
    _, accuracy_test = model.evaluate(x_test, y_test, batch_size=batch_size, verbose=0)
    _, accuracy_train = model.evaluate(x_train, y_train, batch_size=batch_size, verbose=0)

    if score_nowalk is None:
        score_nowalk = accuracy_test
    elif accuracy_test > score_nowalk:
        score_nowalk = accuracy_test
        model_nowalk = model

    print("testing accuracy: ", accuracy_test, "training accuracy: ", accuracy_train)

#    print(precision_recall_fscore_support(y_true, y_pred))
```

```
/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

testing accuracy:  0.9174603223800659 training accuracy:  0.9888446927070618

/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

testing accuracy:  0.9126983880996704 training accuracy:  0.9772909283638

/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

testing accuracy:  0.9539682865142822 training accuracy:  0.9908366799354553

/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

testing accuracy:  0.9327999353408813 training accuracy:  0.988071620464325

/usr/local/lib/python3.7/site-packages/tensorflow_core/python/framework/constant_op.py:96: ComplexWarning: C
asting complex values to real discards the imaginary part
  return ops.EagerTensor(value, ctx.device_name, dtype)

testing accuracy:  0.9248000383377075 training accuracy:  0.9757455587387085
```

In [37]:
```
model_nowalk.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_25 (Conv1D)           (None, 19, 64)            832
_________________________________________________________________
conv1d_26 (Conv1D)           (None, 18, 64)            8256
_________________________________________________________________
dropout_13 (Dropout)         (None, 18, 64)            0
_________________________________________________________________
max_pooling1d_13 (MaxPooling (None, 9, 64)             0
_________________________________________________________________
flatten_13 (Flatten)         (None, 576)               0
_________________________________________________________________
dense_25 (Dense)             (None, 100)               57700
_________________________________________________________________
dense_26 (Dense)             (None, 5)                 505
=================================================================
Total params: 67,293
Trainable params: 67,293
Non-trainable params: 0
_________________________________________________________________
```

4.2.4 Accuracy for ensembled model

In [38]:
```
predictions_cl = model_walk.predict_classes(X_walk)
predictions_nonwalk_cl = model_nowalk.predict_classes(X_walk)
print(predictions_cl.shape, predictions_nonwalk_cl.shape, y.shape)
```

```
(1663,) (1663,) (1663,)
```

In [39]:
```
idx = np.where(predictions_cl == 1)
predictions_cl[idx] = 5
idx = np.where(predictions_cl == 0)
predictions_cl[idx] = predictions_nonwalk_cl[idx]
predictions_cl = np.array(predictions_cl)
y = np.array(y)
error = predictions_cl - y
erridx = np.where(error != 0)
error = len(erridx)/len(y)
print("error=", error, ", accuracy=", 1-error)
```

```
error= 0.0006013229104028864 , accuracy= 0.9993986770895971
```

5. Sources

- [1] https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/ (https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/)
- [2] https://machinelearningmastery.com/deep-learning-models-for-human-activity-recognition/ (https://machinelearningmastery.com/deep-learning-models-for-human-activity-recognition/)
- [3] https://keras.io/optimizers/ (https://keras.io/optimizers/)
- [4] http://cs229.stanford.edu/proj2015/054_report.pdf (http://cs229.stanford.edu/proj2015/054_report.pdf)
- [5] https://keras.io/losses/ (https://keras.io/losses/)