Fast lap analysis

First create an InfluxDB client by importing the modules and setting all the required configuration.

In [43]:

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import influxdb_client
from influxdb_client.client.write_api import SYNCHRONOUS

import warnings
from influxdb_client.client.warnings import MissingPivotFunction

warnings.simplefilter("ignore", MissingPivotFunction)

# configure influxdb client
ORG = "b4mad"
TOKEN = os.environ.get(
    "INFLUXDB_TOKEN",
    "citqAMr66LLb25hvaaZm2LezOc88k2ocOFJcJDR6QB-RmLJa_-sAr9kYB4vSFYaz8bt26lm
)
URL = "https://telemetry.b4mad.racing/"

# and create the client and a quary api
client = influxdb_client.InfluxDBClient(url=URL, token=TOKEN, org=ORG)
query_api = client.query_api()

gameName = "iRacing"
trackCode = "sebring international"
carModel = "Ferrari 488 GT3 Evo 2020"
```

Find all sessions for our track and car.

In [44]:

```
query = f"""
    from(bucket: "racing")
        |> range(start:-10y, stop: now())
        |> filter(fn: (r) => r._field == "CurrentLapTime" and r["GameName"]
        |> filter(fn: (r) => r["CarModel"] == "{carModel}" )
        |> filter(fn: (r) => r["TrackCode"] == "{trackCode}" )
        |> last()
        |> limit(n: 1)
        |> keep(columns: ["_time", "_value", "CarModel","TrackCode", "Sessio
        |> group()
"""

df = query_api.query_data_frame(org=ORG, query=query)
df
```

Out[44]:

	result	table	_time	_value	CarModel	SessionId	SessionTypeI
0	_result	0	2022-11-22 18:56:23.426000+00:00	101.137070	Ferrari 488 GT3 Evo 2020	1669141800	Pr
1	_result	0	2022-11-22 19:00:01.518000+00:00	0.000000	Ferrari 488 GT3 Evo 2020	1669143592	
2	_result	0	2022-11-22 19:00:03.579000+00:00	2.023300	Ferrari 488 GT3 Evo 2020	1669143601	
3	_result	0	2022-11-22 19:01:02.437000+00:00	60.839966	Ferrari 488 GT3 Evo 2020	1669143603	
4	_result	0	2022-11-22 19:01:43.521000+00:00	0.000000	Ferrari 488 GT3 Evo 2020	1669143678	
5	_result	0	2022-11-22 19:01:45.643000+00:00	2.156700	Ferrari 488 GT3 Evo 2020	1669143703	
6	_result	0	2022-11-22 19:06:25.404000+00:00	21.816000	Ferrari 488 GT3 Evo 2020	1669143705	
7	_result	0	2022-11-22 19:06:51.497000+00:00	0.000000	Ferrari 488 GT3 Evo 2020	1669143986	
8	_result	0	2022-11-22 19:06:54.695000+00:00	3.223333	Ferrari 488 GT3 Evo 2020	1669144011	
9	_result	0	2022-11-22 19:10:17.410000+00:00	70.870100	Ferrari 488 GT3 Evo 2020	1669144014	
10	_result	0	2022-11-22 19:10:45.364000+00:00	0.000000	Ferrari 488 GT3 Evo 2020	1669144218	
11	_result	0	2022-11-22 19:10:48.716000+00:00	3.340000	Ferrari 488 GT3 Evo 2020	1669144245	
12	_result	0	2022-11-22 19:11:28.909000+00:00	43.540000	Ferrari 488 GT3 Evo 2020	1669144248	
13	_result	0	2022-11-22 19:11:56.047000+00:00	0.000000	Ferrari 488 GT3 Evo 2020	1669144289	
14	_result	0	2022-11-22 19:11:57.739000+00:00	1.706667	Ferrari 488 GT3 Evo 2020	1669144316	
15	_result	0	2022-11-22 19:16:04.548000+00:00	110.568169	Ferrari 488 GT3 Evo 2020	1669144317	

	result	table	_time	_value	CarModel	SessionId	SessionTypel
16	_result	0	2022-11-22 19:16:31.717000+00:00	0.000000	Ferrari 488 GT3 Evo 2020	1669144565	
17	_result	0	2022-11-22 19:16:33.778000+00:00	2.040000	Ferrari 488 GT3 Evo 2020	1669144591	
18	_result	0	2022-11-22 19:20:34.854000+00:00	108.409600	Ferrari 488 GT3 Evo 2020	1669144593	
19	_result	0	2022-11-22 19:21:01.162000+00:00	0.000000	Ferrari 488 GT3 Evo 2020	1669144835	
20	_result	0	2022-11-22 19:21:03.807000+00:00	2.640000	Ferrari 488 GT3 Evo 2020	1669144861	
21	_result	0	2022-11-22 19:25:46.995000+00:00	149.915665	Ferrari 488 GT3 Evo 2020	1669144863	
22	_result	0	2022-11-22 19:26:13.734000+00:00	0.000000	Ferrari 488 GT3 Evo 2020	1669145147	
23	_result	0	2022-11-22 19:26:15.833000+00:00	2.090000	Ferrari 488 GT3 Evo 2020	1669145173	
24	_result	0	2022-11-22 19:26:48.959000+00:00	35.190000	Ferrari 488 GT3 Evo 2020	1669145175	
25	_result	0	2022-11-22 19:27:16.451000+00:00	0.000000	Ferrari 488 GT3 Evo 2020	1669145209	
26	_result	0	2022-11-22 19:27:19.865000+00:00	3.389967	Ferrari 488 GT3 Evo 2020	1669145236	
27	_result	0	2022-11-22 19:29:12.062000+00:00	115.556633	Ferrari 488 GT3 Evo 2020	1669145239	
28	_result	0	2022-11-22 19:29:39.616000+00:00	0.000000	Ferrari 488 GT3 Evo 2020	1669145353	
29	_result	0	2022-11-22 19:29:42.906000+00:00	3.256667	Ferrari 488 GT3 Evo 2020	1669145379	
30	_result	0	2022-11-22 19:39:14.741000+00:00	60.394535	Ferrari 488 GT3 Evo 2020	1669145383	
31	_result	0	2022-11-23 18:46:05.826000+00:00	219.805161	Ferrari 488 GT3 Evo 2020	1669227579	Pr

	result	table	_time	_value	CarModel	SessionId	SessionType
32	_result	0	2022-11-23 18:48:37.124000+00:00	0.000000	Ferrari 488 GT3 Evo 2020	1669229229	Pra
33	_result	0	2022-11-23 18:56:22.547000+00:00	132.544968	Ferrari 488 GT3 Evo 2020	1669229318	Q
34	_result	0	2022-11-23 19:19:00.245000+00:00	194.217529	Ferrari 488 GT3 Evo 2020	1669230142	
35	_result	0	2022-11-23 19:36:41.490000+00:00	0.000000	Ferrari 488 GT3 Evo 2020	1669231908	Pra
36	_result	0	2022-11-23 19:45:54.959000+00:00	0.000000	Ferrari 488 GT3 Evo 2020	1669232292	Pra
37	_result	0	2022-11-23 19:56:11.593000+00:00	3.566733	Ferrari 488 GT3 Evo 2020	1669232937	Q
38	_result	0	2022-11-23 20:01:12.899000+00:00	0.000000	Ferrari 488 GT3 Evo 2020	1669233401	
39	_result	0	2022-11-23 20:17:36.039000+00:00	116.288500	Ferrari 488 GT3 Evo 2020	1669233672	

Now we can query the data for a specific session.

In [45]:

```
# get the last row in the dataframe
SESSION = df.iloc[-1]["SessionId"]

query = f"""
from(bucket: "racing")
  |> range(start: -10y, stop: now())
  |> filter(fn: (r) => r["_measurement"] == "laps_cc")
//  |> filter(fn: (r) => r["_field"] == "DistanceRoundTrack" or r["_field"]
  |> filter(fn: (r) => r["SessionId"] == "{SESSION}")
  |> pivot(rowKey: ["_time"], columnKey: ["_field"], valueColumn: "_value")
  |> sort(columns: ["_time"], desc: false)
"""

df = query_api.query_data_frame(org=ORG, query=query)
df
```

Out[45]:

	result	table	_start	_stop	_time
0	_result	0	2012-11-23 23:06:04.927784+00:00	2022-11-24 11:06:04.927784+00:00	2022-11-23 20:01:12.930000+00:00
1	_result	0	2012-11-23 23:06:04.927784+00:00	2022-11-24 11:06:04.927784+00:00	2022-11-23 20:01:12.961000+00:00
2	_result	0	2012-11-23 23:06:04.927784+00:00	2022-11-24 11:06:04.927784+00:00	2022-11-23 20:01:12.992000+00:00
3	_result	0	2012-11-23 23:06:04.927784+00:00	2022-11-24 11:06:04.927784+00:00	2022-11-23 20:01:13.022000+00:00
4	_result	0	2012-11-23 23:06:04.927784+00:00	2022-11-24 11:06:04.927784+00:00	2022-11-23 20:01:13.053000+00:00
...	...	...	...	...	...
31696	_result	0	2012-11-23 23:06:04.927784+00:00	2022-11-24 11:06:04.927784+00:00	2022-11-23 20:17:35.916000+00:00
31697	_result	0	2012-11-23 23:06:04.927784+00:00	2022-11-24 11:06:04.927784+00:00	2022-11-23 20:17:35.947000+00:00
31698	_result	0	2012-11-23 23:06:04.927784+00:00	2022-11-24 11:06:04.927784+00:00	2022-11-23 20:17:35.978000+00:00
31699	_result	0	2012-11-23 23:06:04.927784+00:00	2022-11-24 11:06:04.927784+00:00	2022-11-23 20:17:36.009000+00:00
31700	_result	0	2012-11-23 23:06:04.927784+00:00	2022-11-24 11:06:04.927784+00:00	2022-11-23 20:17:36.039000+00:00

31701 rows × 25 columns

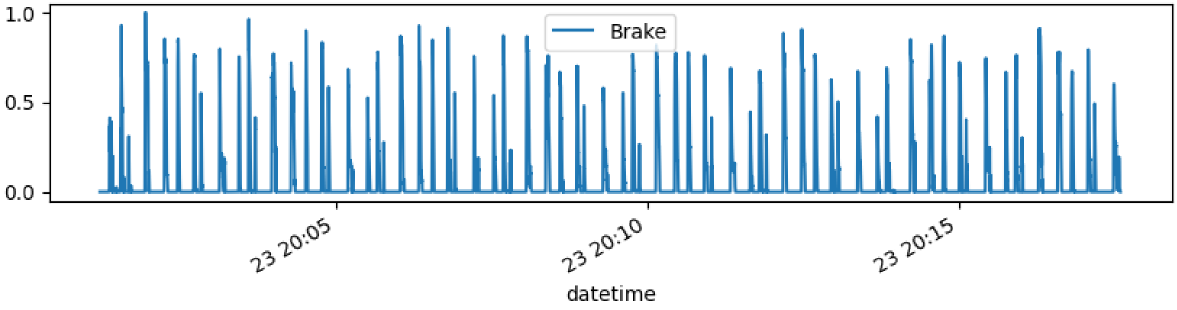
Now we can plot the data, starting with just the brake inputs and the time on the x axis.

In [46]:

```
brake = df.copy()
brake = brake[["Brake", "_time"]]
brake["datetime"] = pd.to_datetime(brake["_time"])
brake.drop(columns=["_time"], inplace=True)
brake.set_index("datetime", inplace=True)
brake.sort_index(inplace=True)
plt.rcParams["figure.figsize"] = (10, 2)
brake.plot()
```

Out[46]:

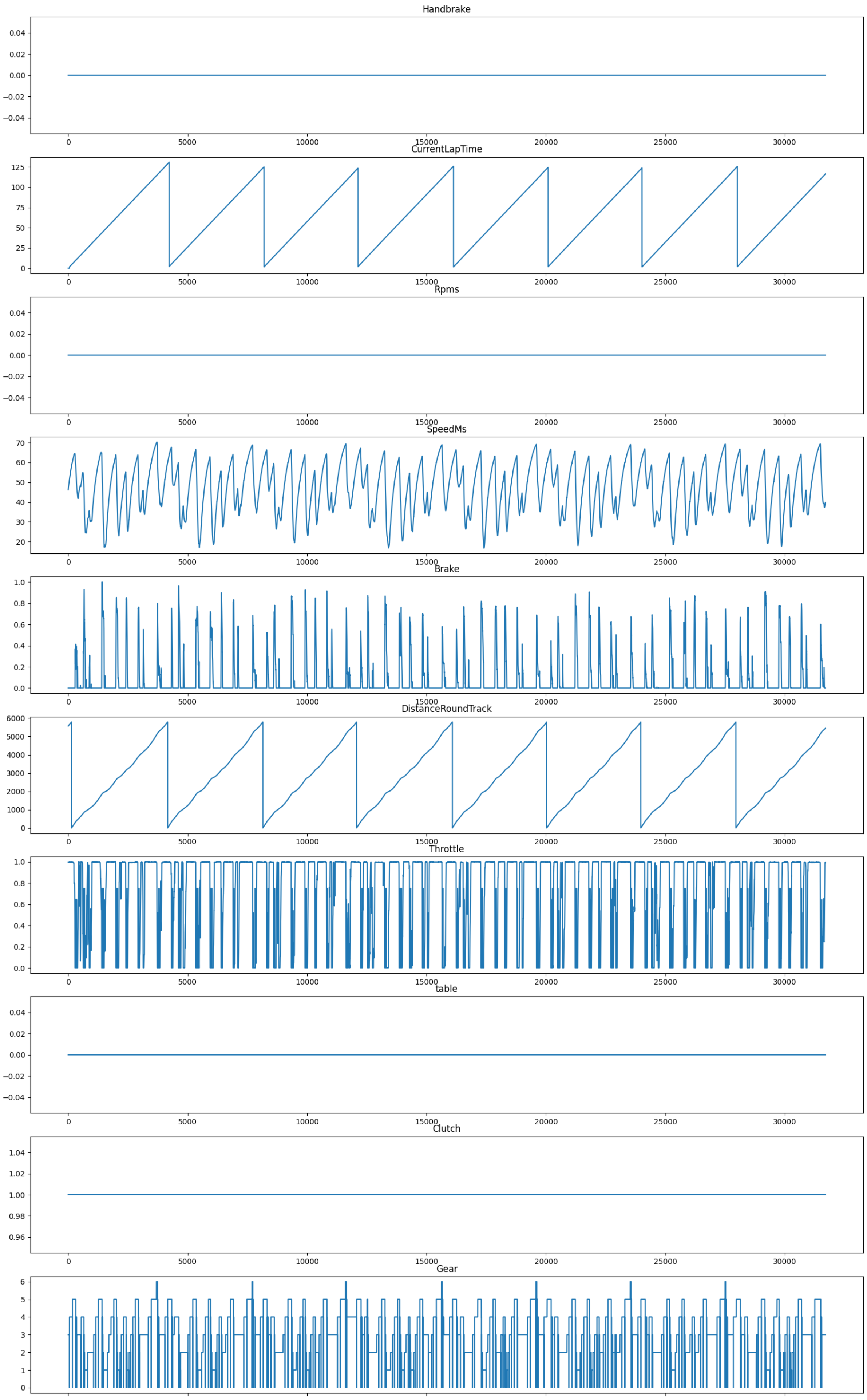
```
<AxesSubplot:xlabel='datetime'>
```

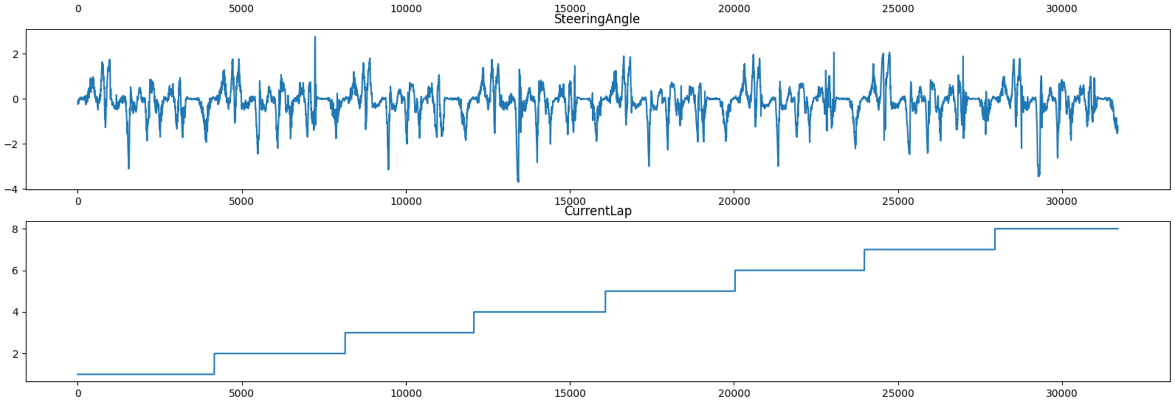


Now plot every other value column, the x-axis is the just the index of the DataFrame.

In [47]:

```
plt.rcParams["figure.figsize"] = (20, 40)
numerics = ["int16", "int32", "int64", "float16", "float32", "float64"]
newdf = df.select_dtypes(include=numerics)
cols = set(newdf.columns)
fig, ax = plt.subplots(len(cols))
for i, c in enumerate(cols):
    newdf[c].astype(float).plot(ax=ax[i])
    ax[i].set_title(c)
plt.show()
```

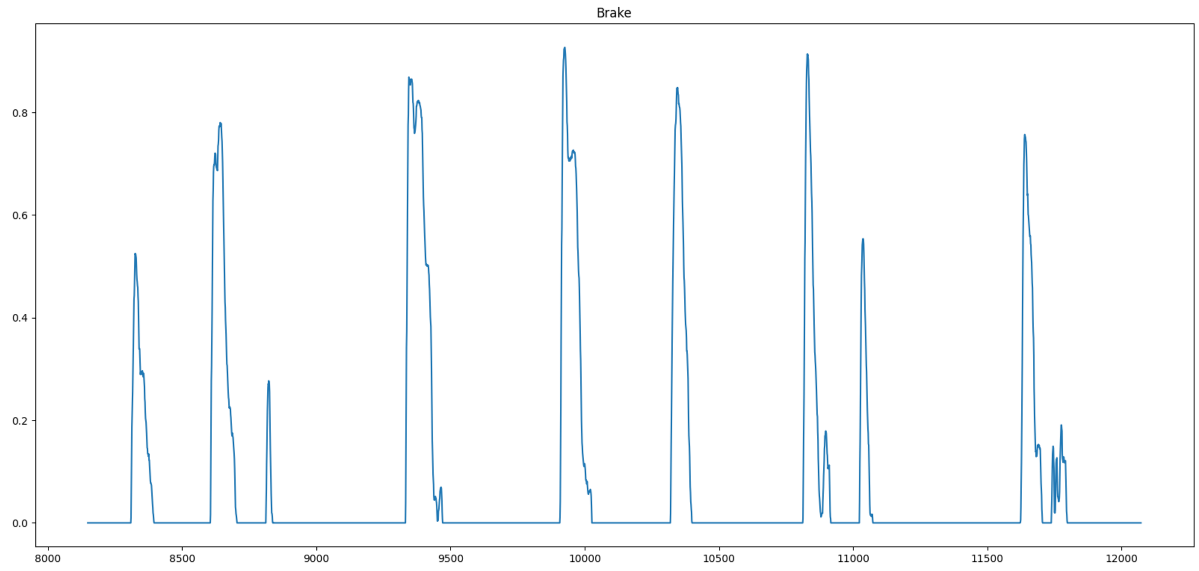

Handbrake
0.04
0.02
0.00
−0.02
−0.04
0
5000
10000
15000
20000
25000
30000
CurrentLapTime
125
100
75
50
25
0
0
5000
10000
15000
20000
25000
30000
Rpms
0.04
0.02
0.00
−0.02
−0.04
0
5000
10000
15000
20000
25000
30000
SpeedMs
70
60
50
40
30
20
0
5000
10000
15000
20000
25000
30000
Brake
1.0
0.8
0.6
0.4
0.2
0.0
0
5000
10000
15000
20000
25000
30000
DistanceRoundTrack
6000
5000
4000
3000
2000
1000
0
0
5000
10000
15000
20000
25000
30000
Throttle
1.0
0.8
0.6
0.4
0.2
0.0
0
5000
10000
15000
20000
25000
30000
table
0.04
0.02
0.00
−0.02
−0.04
0
5000
10000
15000
20000
25000
30000
Clutch
1.04
1.02
1.00
0.98
0.96
0
5000
10000
15000
20000
25000
30000
Gear
6
5
4
3
2
1
0

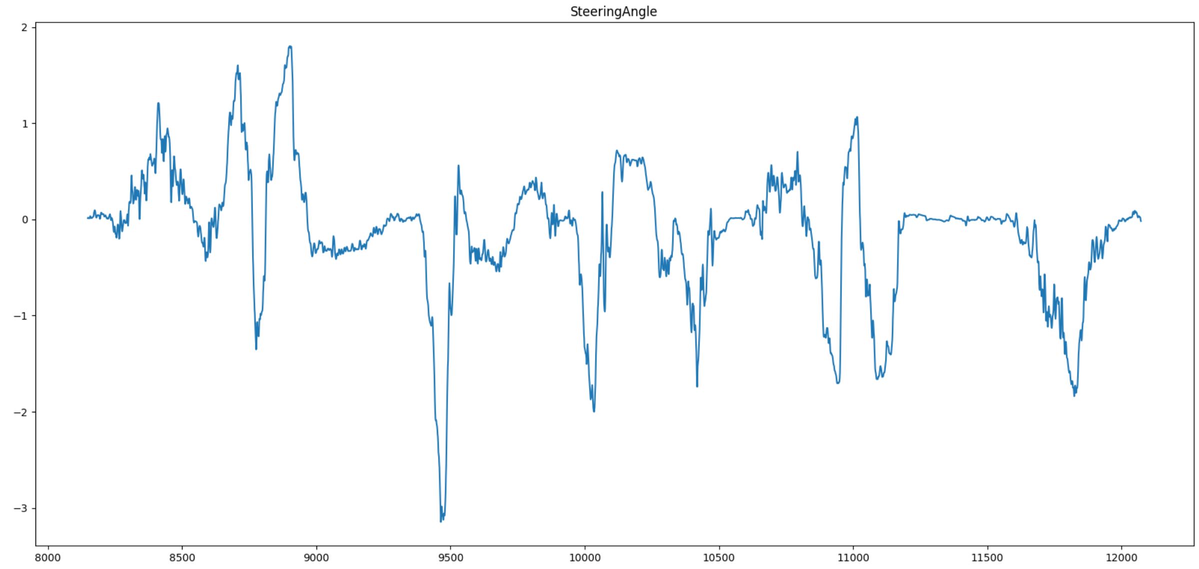


And just a single lap.

In [48]:

```
lap = 3
plt.rcParams["figure.figsize"] = (20, 20)
cols = ["Brake", "SteeringAngle"]
fig, ax = plt.subplots(len(cols))
for i, c in enumerate(cols):
    df.loc[df["CurrentLap"] == lap, c].astype(float).plot(ax=ax[i])
    ax[i].set_title(c)
plt.show()
```

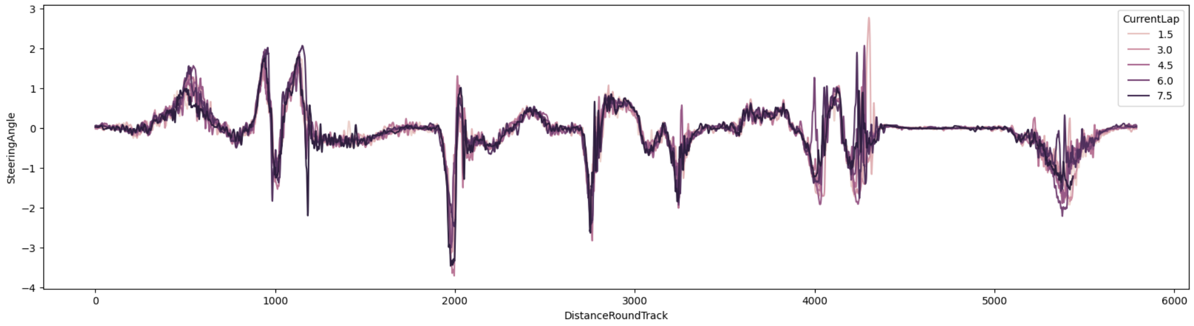


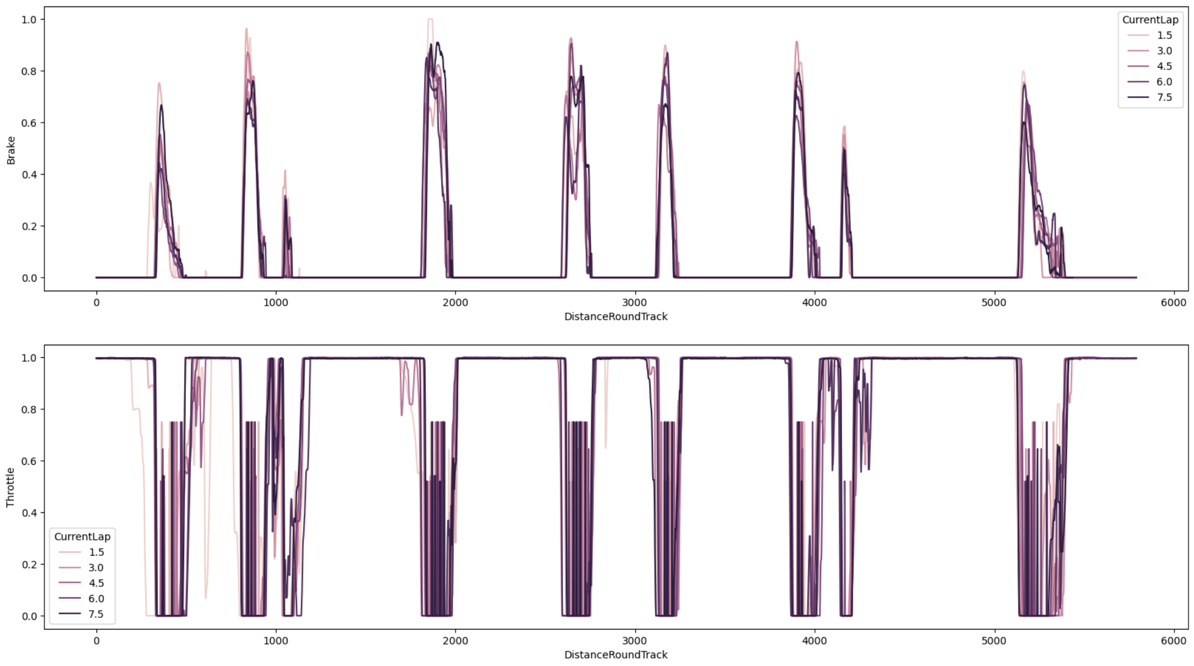


Now we plot all brake values for all laps against the distance.

In [49]:

```
plt.rcParams["figure.figsize"] = (20, 5)
for c in ["SteeringAngle", "Brake", "Throttle"]:
    sns.lineplot(data=df, x="DistanceRoundTrack", y=c, hue="CurrentLap", leg
    plt.show()
```





Find n fastest laps

Iterate over all sessions and find the n fastest laps. A lap is considered complete, if the `DistanceRoundTrack` is reached. The time for a lap is the last value of the `CurrentLapTime` column.

Splice a track into segments

Using the combined / averaged values of the `SteeringAngle` data, we want to splice the track `DistanceRoundTrack` into segments.

- A segement is defined by the start and end of a major turn.
- Each segment connects directly to the next segment.
- The start of a segment is the middle betwee two turns (i.e not just at the beginning of a turn).

See below for the example of the `sebring international` track.

Extract track data from fastest laps

From all fastest laps we want to extract the average value for the track guide data.

In [50]:

```
# load csv into dataframe
df = pd.read_csv("../pitcrew/Ferrari 488 GT3 Evo 2020-sebring international.
df
```

Out[50]:

	turn	start	end	brake	turn_in	force	gear	speed	accelerate	mark
0	1	0	750	325.0	325.0	50.0	4.0	180.0	510.0	NaN
1	2-4	750	1000	825.0	875.0	80.0	2.0	100.0	950.0	NaN
2	5	1000	1250	1040.0	1040.0	40.0	2.0	110.0	1100.0	NaN
3	6	1250	1800	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	7	1800	2100	1825.0	1930.0	80.0	1.0	70.0	1975.0	NaN
5	8-9	2100	2400	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	10-11	2400	3000	2640.0	2700.0	90.0	2.0	90.0	2750.0	NaN
7	12-13	3000	3350	3130.0	3180.0	80.0	2.0	110.0	3230.0	NaN
8	14	3350	3750	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	15	3750	4050	3870.0	3920.0	80.0	3.0	130.0	4000.0	NaN
10	16	4050	4300	4150.0	4150.0	40.0	3.0	160.0	4230.0	NaN
11	17	4300	5750	5150.0	5250.0	80.0	3.0	130.0	5350.0	NaN

- start / end: the start and end of the turn (see above)
- brake: the average DistanceRoundTrack when the brake is pressed the first time
- turn_in: the average DistanceRoundTrack when the steering wheel is turned into the corner (maybe use rate of change)
- force: the average value of the maximum brake force during the turn
- gear: the average value of the lowest gear during the turn
- speed: the lowest value during the turn
- stop: the average value when the brake force is starting to decrease
- accelerate: the average DistanceRoundTrack when the throttle is pressed again during the turn