Training_analysis

October 30, 2022

1 Training analysis for DeepRacer

This notebook has been built based on the `DeepRacer Log Analysis.ipynb` provided by the AWS DeepRacer Team. It has been reorganised and expanded to provide new views on the training data without the helper code which was moved into utility `.py` files.

1.1 Usage

I have expanded this notebook from to present how I'm using this information. It contains descriptions that you may find not that needed after initial reading. Since this file can change in the future, I recommend that you make its copy and reorganize it to your liking. This way you will not lose your changes and you'll be able to add things as you please.

This notebook isn't complete. What I find interesting in the logs may not be what you will find interesting and useful. I recommend you get familiar with the tools and try hacking around to get the insights that suit your needs.

1.2 Contributions

As usual, your ideas are very welcome and encouraged so if you have any suggestions either bring them to the AWS DeepRacer Community or share as code contributions.

1.3 Training environments

Depending on whether you're running your training through the console or using the local setup, and on which setup for local training you're using, your experience will vary. As much as I would like everything to be taylored to your configuration, there may be some problems that you may face. If so, please get in touch through the AWS DeepRacer Community.

1.4 Requirements

Before you start using the notebook, you will need to install some dependencies. If you haven't yet done so, have a look at The README.md file to find what you need to install.

Apart from the install, you also have to configure your programmatic access to AWS. Have a look at the guides below, AWS resources will lead you by the hand:

AWS CLI: https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html

Boto Configuration: https://boto3.amazonaws.com/v1/documentation/api/latest/guide/configuration.html

1.5 Credits

I would like to thank the AWS DeepRacer Community for all the feedback about the notebooks. If you'd like, follow my blog where I tend to write about my experiences with AWS DeepRacer.

2 Log Analysis

Let's get to it.

2.1 Permissions

Depending on where you are downloading the data from, you will need some permissions: * Access to CloudWatch log streams * Access to S3 bucket to reach the log files

2.2 Installs and setups

If you are using an AWS SageMaker Notebook to run the log analysis, you will need to ensure you install required dependencies. To do that uncomment and run the following:

[1]:
```
# Make sure you have deepracer-utils >= 0.9

# import sys

# !{sys.executable} -m pip install --upgrade deepracer-utils
```

2.3 Imports

Run the imports block below:

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from pprint import pprint

from deepracer.tracks import TrackIO, Track
from deepracer.tracks.track_utils import track_breakdown, track_meta
from deepracer.logs import \
    SimulationLogsIO as slio, \
    NewRewardUtils as nr, \
    AnalysisUtils as au, \
    PlottingUtils as pu, \
    ActionBreakdownUtils as abu, \
    DeepRacerLog

# Ignore deprecation warnings we have no power over
import warnings
warnings.filterwarnings('ignore')
```

2.4 Get the logs

Depending on which way you are training your model, you will need a slightly different way to load the data.

AWS DeepRacer Console

The logs can be downloaded from the training page. Once you download them, extract the archive into logs/[training-name] (just like logs/sample-logs)

DeepRacer for Cloud

If you're using local training, just point at your model's root folder in the minio bucket. If you're using any of the cloudy deployments, download the model folder to local and point at it.

Deepracer for dummies/Chris Rhodes' Deepracer/ARCC Deepracer or any training solution other than the ones above, read below

This notebook has been updated to support the most recent setups. Most of the mentioned projects above are no longer compatible with AWS DeepRacer Console anyway so do consider moving to the ones actively maintained.

[3]:
```
model_logs_root = 'Practice-2-clone-clone-qualifier/train'
log = DeepRacerLog(model_logs_root)

# load logs into a dataframe
log.load_robomaker_logs()

try:
    pprint(log.agent_and_network())
    print("-------------")
    pprint(log.hyperparameters())
    print("-------------")
    pprint(log.action_space())
except Exception:
    print("Robomaker logs not available")

df = log.dataframe()
```

```
{'network': 'DEEP_CONVOLUTIONAL_NETWORK_SHALLOW',
 'sensor_list': ['FRONT_FACING_CAMERA'],
 'simapp_version': '5.0',
 'world': 'reInvent2019_track'}
-------------
{'batch_size': 64,
 'beta_entropy': 0.01,
 'discount_factor': 0.99,
 'e_greedy_value': 1.0,
 'epsilon_steps': 10000,
 'exploration_type': 'categorical',
 'loss_type': 'huber',
```

```
 'lr': 0.0006,
 'num_episodes_between_training': 10,
 'num_epochs': 3,
 'stack_size': 1,
 'term_cond_avg_score': 100000.0,
 'term_cond_max_episodes': 100000}
-------------
{'speed': {'high': 4, 'low': 0.6}, 'steering_angle': {'high': 30, 'low': -15}}
```

If the code above worked, you will see a list of details printed above: a bit about the agent and the network, a bit about the hyperparameters and some information about the action space. Now let's see what got loaded into the dataframe - the data structure holding your simulation information. the `head()` method prints out a few first lines of the data:

[4]:
```
df.head()
```

[4]:
```
   iteration  episode  steps       x       y      yaw  steering_angle  speed  \
0          1        0      3  0.3278  2.6829 -83.0066           19.45   0.60
1          1        0      4  0.3393  2.6633 -81.1855           30.00   2.21
2          1        0      5  0.3617  2.6323 -77.0699            9.57   4.00
3          1        0      6  0.3931  2.5893 -72.3723          -15.00   4.00
4          1        0      7  0.4177  2.5318 -70.9445           30.00   4.00

   action    reward  done on_track  progress  closest_waypoint  track_len  \
0      -1    0.0010     0     True    0.6446                 1      23.12
1      -1   72.0178     0     True    0.7360                 1      23.12
2      -1  104.3333     0     True    0.8826                 1      23.12
3      -1  104.2000     0     True    1.0856                 2      23.12
4      -1  104.0667     0     True    1.3609                 2      23.12

   tstamp episode_status  pause_duration
0  20.924    in_progress             0.0
1  20.986    in_progress             0.0
2   21.05    in_progress             0.0
3  21.079    in_progress             0.0
4  21.176    in_progress             0.0
```

2.5 Load waypoints for the track you want to run analysis on

The track waypoint files represent the coordinates of characteristic points of the track - the center line, inside border and outside border. Their main purpose is to visualise the track in images below.

The naming of the tracks is not super consistent. The ones that we already know have been mapped to their official names in the track_meta dictionary.

Some npy files have an 'Eval' suffix. One of the challenges in the past was that the evaluation tracks were different to physical tracks and we have recreated them to enable evaluation. Remeber that evaluation npy files are a community effort to visualise the tracks in the trainings, they aren't 100% accurate.

Tracks Available:

```
[6]: tu = TrackIO()

for track in tu.get_tracks():
    print("{} - {}".format(track, track_meta.get(track[:-4], "I don't know")))
```

```
2022_april_open.npy - I don't know
2022_april_pro.npy - I don't know
2022_august_open.npy - I don't know
2022_august_pro.npy - I don't know
2022_july_open.npy - I don't know
2022_july_pro.npy - I don't know
2022_june_open.npy - I don't know
2022_june_pro.npy - I don't know
2022_march_open.npy - I don't know
2022_march_pro.npy - I don't know
2022_may_open.npy - I don't know
2022_may_pro.npy - I don't know
2022_october_open.npy - I don't know
2022_october_pro.npy - I don't know
2022_reinvent_champ.npy - I don't know
2022_september_open.npy - I don't know
2022_september_pro.npy - I don't know
2022_summit_speedway.npy - I don't know
2022_summit_speedway_mini.npy - I don't know
AWS_track.npy - I don't know
Albert.npy - Yun Speedway
AmericasGeneratedInclStart.npy - Badaal Track
Aragon.npy - Stratus Loop
Austin.npy - American Hills Speedway
Belille.npy - Cumulo Turnpike
Bowtie_track.npy - Bowtie Track
Canada_Eval.npy - Toronto Turnpike Eval
Canada_Training.npy - Toronto Turnpike Training
China_eval_track.npy - Shanghai Sudu Eval
China_track.npy - Shanghai Sudu Training
FS_June2020.npy - Fumiaki Loop
H_track.npy - H track
July_2020.npy - Roger Raceway
LGSWide.npy - SOLA Speedway
London_Loop_Train.npy - I don't know
Mexico_track.npy - Cumulo Carrera Training
Mexico_track_eval.npy - Cumulo Carrera Eval
Monaco.npy - European Seaside Circuit
New_York_Eval_Track.npy - Empire City Eval
New_York_Track.npy - Empire City Training
Oval_track.npy - Oval Track
```

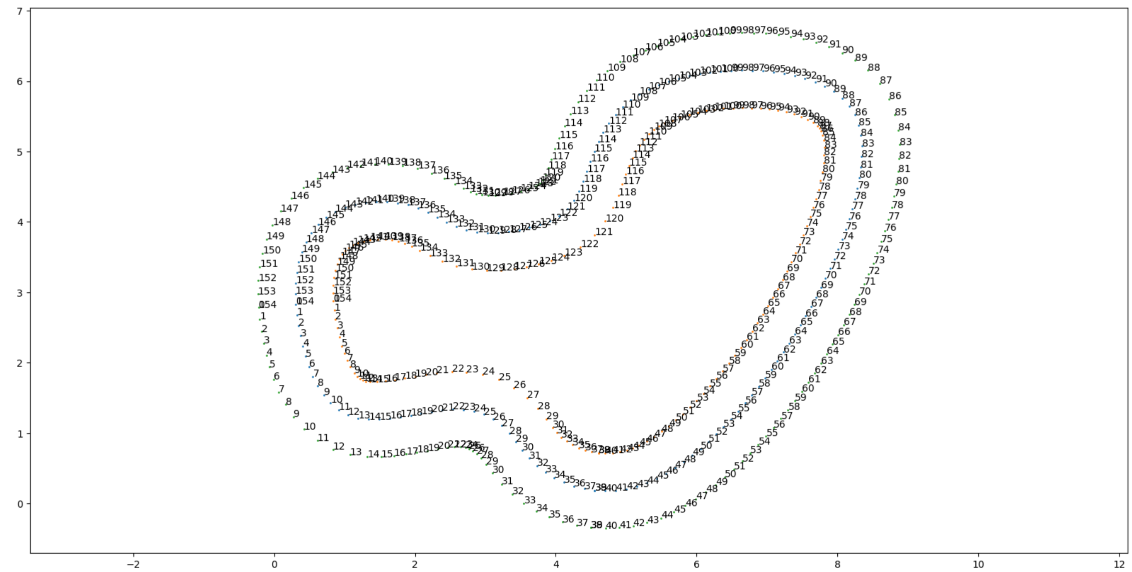
```
Singapore.npy - Asia Pacific Bay Loop
Spain_track.npy - Circuit de Barcelona-Catalunya
Straight_track.npy - Straight track
Tokyo_Training_track.npy - Kumo Torakku Training
Vegas_track.npy - AWS Summit Raceway
Virtual_May19_Train_track.npy - London Loop Training
arctic_open.npy - I don't know
arctic_pro.npy - I don't know
caecer_gp.npy - I don't know
caecer_loop.npy - I don't know
dubai_open.npy - I don't know
dubai_pro.npy - I don't know
hamption_open.npy - I don't know
hamption_pro.npy - I don't know
jyllandsringen_open.npy - I don't know
jyllandsringen_pro.npy - I don't know
morgan_open.npy - I don't know
morgan_pro.npy - I don't know
penbay_open.npy - I don't know
penbay_pro.npy - I don't know
reInvent2019_track.npy - The 2019 DeepRacer Championship Cup
reInvent2019_wide.npy - re:Invent 2018 Wide
reInvent2019_wide_mirrored.npy - re:Invent 2018 Wide Mirrored
red_star_open.npy - I don't know
red_star_pro.npy - I don't know
reinvent_base.npy - re:Invent 2018
thunder_hill_open.npy - I don't know
thunder_hill_pro.npy - I don't know
```

Now let's load the track:

[7]:
```
# We will try to guess the track name first, if it
# fails, we'll use the constant in quotes

try:
    track_name = log.agent_and_network()["world"]
except Exception as e:
    track_name = "reInvent2019_track"


track: Track = tu.load_track(track_name)

pu.plot_trackpoints(track)
```

```
Loaded 155 waypoints
```

[7]:
```
<AxesSubplot: >
```



2.6 Graphs

The original notebook has provided some great ideas on what could be visualised in the graphs. Below examples are a slightly extended version. Let's have a look at what they are presenting and what this may mean to your training.

2.6.1 Training progress

As you have possibly noticed by now, training episodes are grouped into iterations and this notebook also reflects it. What also marks it are checkpoints in the training. After each iteration a set of ckpt files is generated - they contain outcomes of the training, then a model.pb file is built based on that and the car begins a new iteration. Looking at the data grouped by iterations may lead you to a conclusion, that some earlier checkpoint would be a better start for a new training. While this is limited in the AWS DeepRacer Console, with enough disk space you can keep all the checkpoints along the way and use one of them as a start for new training (or even as a submission to a race).

While the episodes in a given iteration are a mixture of decision process and random guesses, mean results per iteration may show a specific trend. Mean values are accompanied by standard deviation to show the concentration of values around the mean.

Rewards per Iteration You can see these values as lines or dots per episode in the AWS DeepRacer console. When the reward goes up, this suggests that a car is learning and improving with regards to a given reward function. **This does not have to be a good thing.** If your reward function rewards something that harms performance, your car will learn to drive in a way that will make results worse.

At first the rewards just grow if the progress achieved grows. Interesting things may happen slightly later in the training:

- The reward may go flat at some level - it might mean that the car can't get any better. If you think you could still squeeze something better out of it, review the car's progress and consider updating the reward function, the action space, maybe hyperparameters, or perhaps starting over (either from scratch or from some previous checkpoint)
- The reward may become wobbly - here you will see it as a mesh of dots zig-zagging. It can be a gradually growing zig-zag or a roughly stagnated one. This usually means the learning rate hyperparameter is too high and the car started doing actions that oscilate around some local extreme. You can lower the learning rate and hope to step closer to the extreme. Or run away from it if you don't like it
- The reward plunges to near zero and stays roughly flat - I only had that when I messed up the hyperparameters or the reward function. Review recent changes and start training over or consider starting from scratch

The Standard deviation says how close from each other the reward values per episode in a given iteration are. If your model becomes reasonably stable and worst performances become better, at some point the standard deviation may flat out or even decrease. That said, higher speeds usually mean there will be areas on track with higher risk of failure. This may bring the value of standard deviation to a higher value and regardless of whether you like it or not, you need to accept it as a part of fighting for significantly better times.

Time per iteration I'm not sure how useful this graph is. I would worry if it looked very similar to the reward graph - this could suggest that slower laps will be getting higher rewards. But there is a better graph for spotting that below.

Progress per Iteration This graph usually starts low and grows and at some point it will get flatter. The maximum value for progress is 100% so it cannot grow without limits. It usually shows similar initial behaviours to reward and time graphs. I usually look at it when I alter an action in training. In such cases this graph usually dips a bit and then returns or goes higher.

Total reward per episode This graph has been taken from the orignal notebook and can show progress on certain groups of behaviours. It usually forms something like a triangle, sometimes you can see a clear line of progress that shows some new way has been first taught and then perfected.

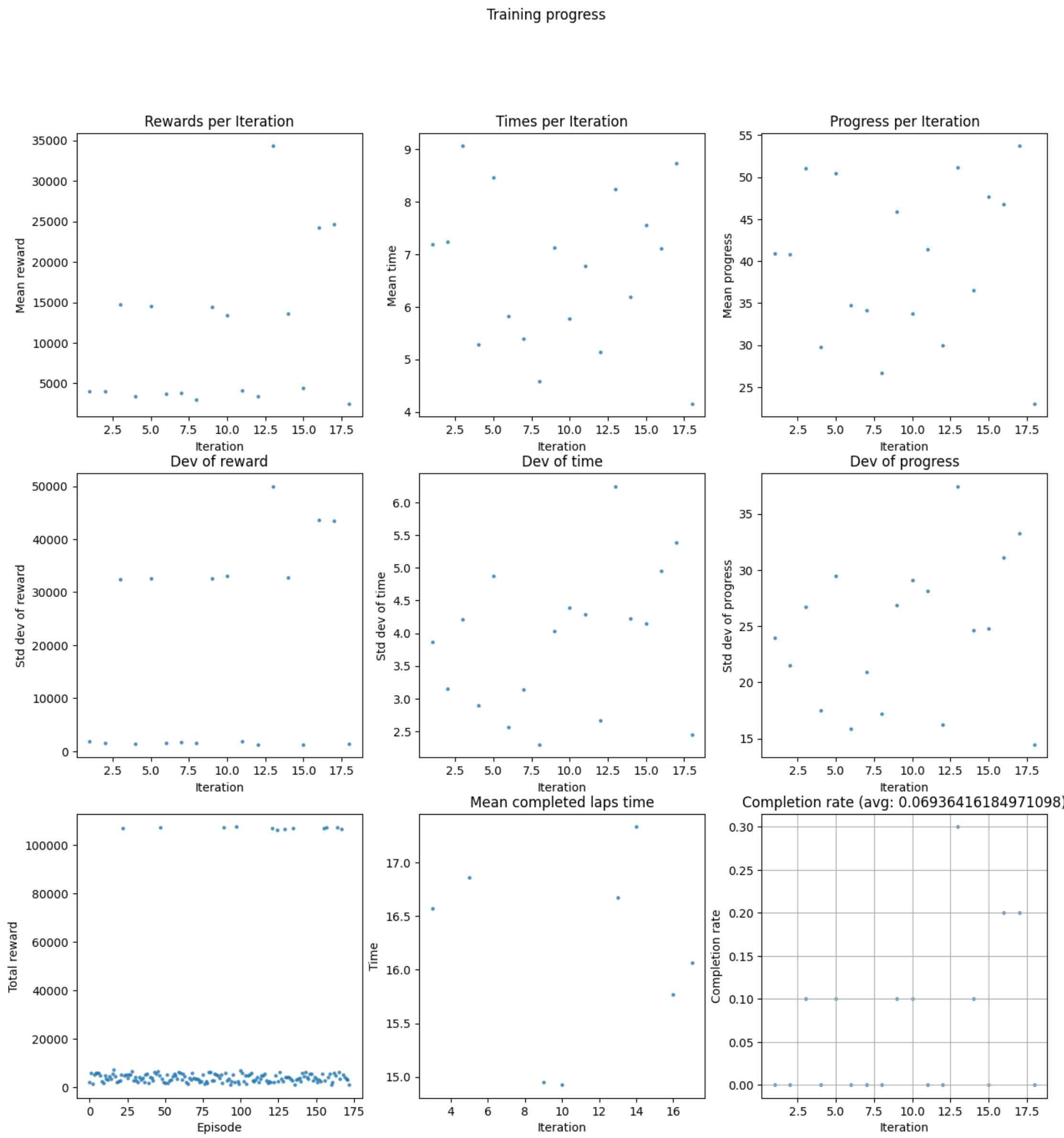
Mean completed lap times per iteration Once we have a model that completes laps reasonably often, we might want to know how fast the car gets around the track. This graph will show you that. I use it quite often when looking for a model to shave a couple more miliseconds. That said it has to go in pair with the last one:

Completion rate per iteration It represents how big part of all episodes in an iteration is full laps. The value is from range [0, 1] and is a result of deviding amount of full laps in iteration by amount of all episodes in iteration. I say it has to go in pair with the previous one because you not only need a fast lapper, you also want a race completer.

The higher the value, the more stable the model is on a given track.

[8]:
```
simulation_agg = au.simulation_agg(df)

au.analyze_training_progress(simulation_agg, title='Training progress')
```

```
new reward not found, using reward as its values
Number of episodes =  172
Number of iterations =  18
```




```
<Figure size 640x480 with 0 Axes>
```

2.6.2 Stats for all laps

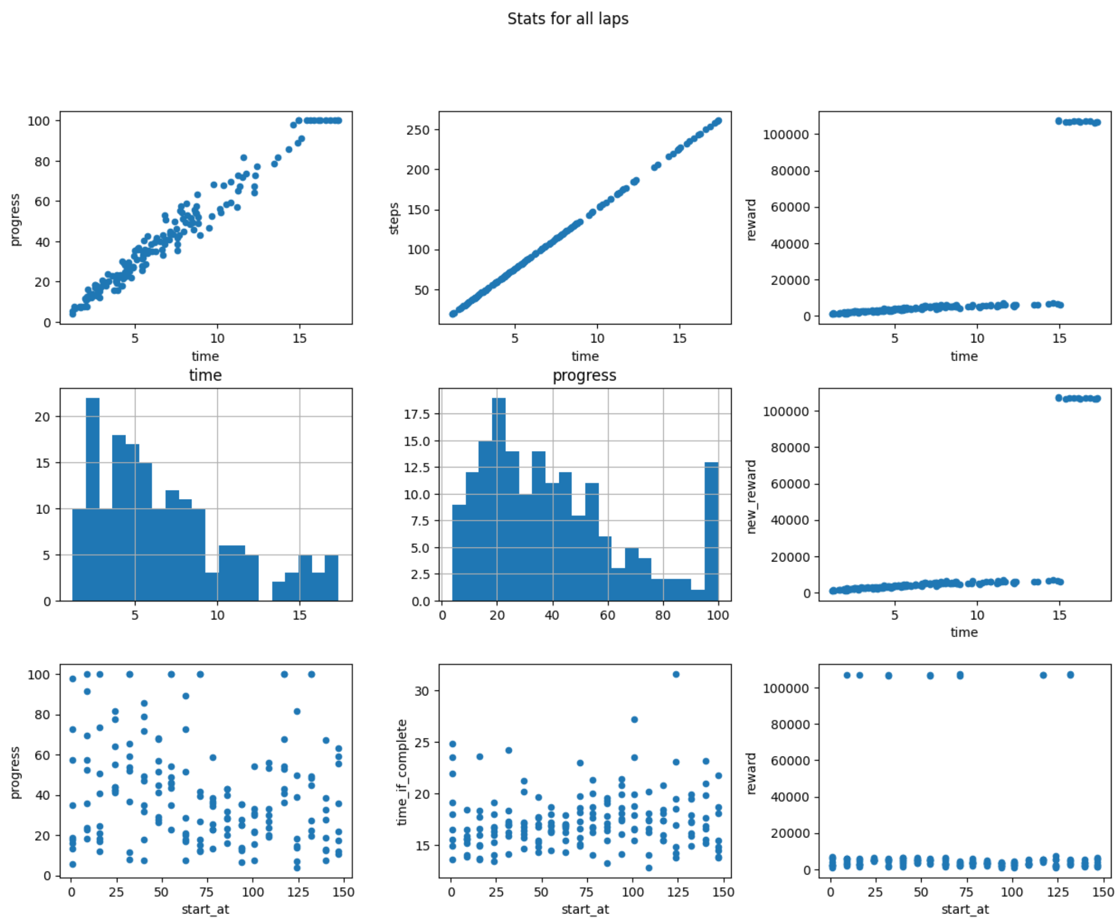
Previous graphs were mainly focused on the state of training with regards to training progress. This however will not give you a lot of information about how well your reward function is doing overall.

In such case `scatter_aggregates` may come handy. It comes with three types of graphs:

* progress/steps/reward depending on the time of an episode - of this I find reward/time and new_reward/time especially useful to see that I am rewarding good behaviours - I expect the reward to time scatter to look roughly triangular * histograms of time and progress - for all episodes the progress one is usually quite handy to get an idea of model's stability * progress/time_if_complete/reward to closest waypoint at start - these are really useful during training as they show potentially problematic spots on track. It can turn out that a car gets best reward (and performance) starting at a point that just cannot be reached if the car starts elsewhere, or that there is a section of a track that the car struggles to get past and perhaps it's caused by an aggressive action space or undesirable behaviour prior to that place

Side note: `time_if_complete` is not very accurate and will almost always look better for episodes closer to 100% progress than in case of those 50% and below.

[9]:
```python
au.scatter_aggregates(simulation_agg, 'Stats for all laps')
```

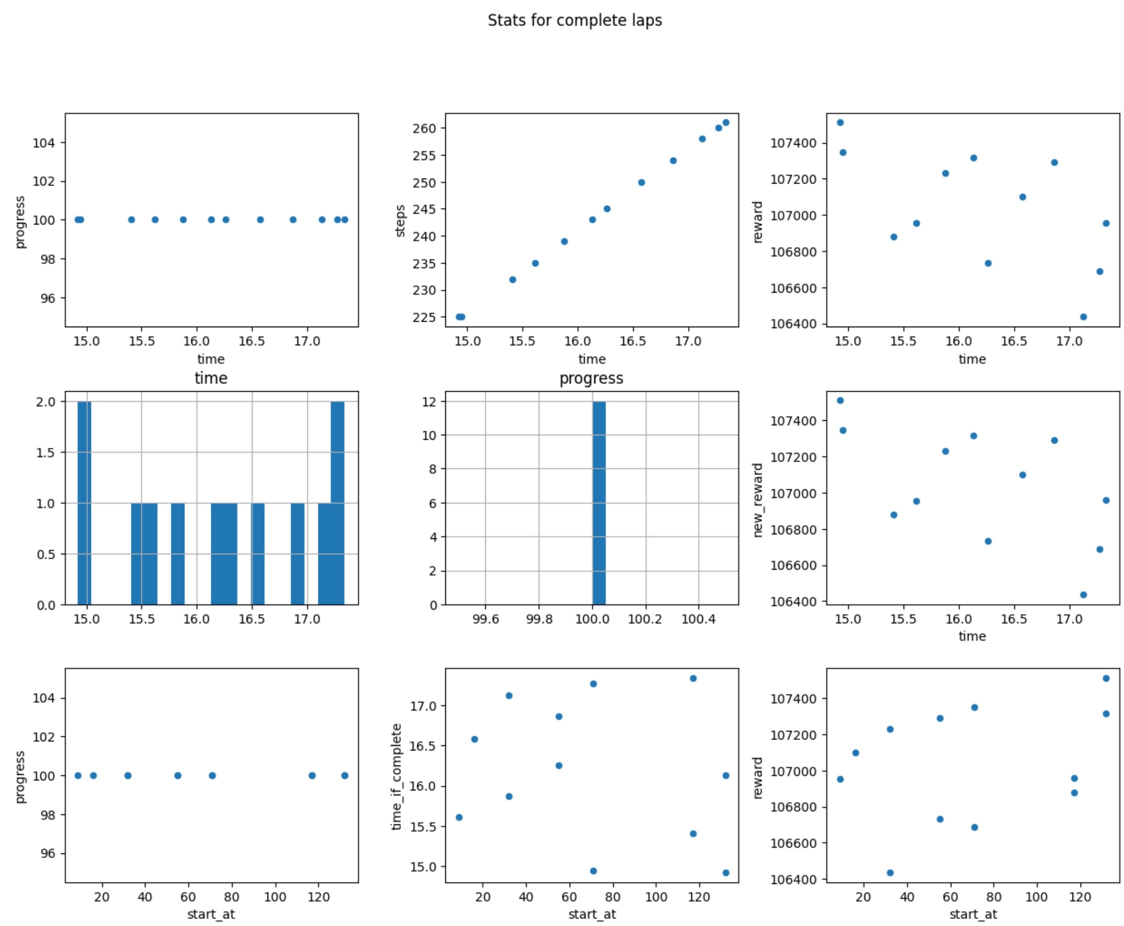



```
<Figure size 640x480 with 0 Axes>
```

2.6.3 Stats for complete laps

The graphs here are same as above, but now I am interested in other type of information: * does the reward scatter show higher rewards for lower completion times? If I give higher reward for a slower lap it might suggest that I am training the car to go slow * what does the time histogram look like? With enough samples available the histogram takes a normal distribution graph shape. The lower the mean value, the better the chance to complete a fast lap consistently. The longer the tails, the greater the chance of getting lucky in submissions * is the car completing laps around the place where the race lap starts? Or does it only succeed if it starts in a place different to the racing one?

[10]:
```
complete_ones = simulation_agg[simulation_agg['progress']==100]

if complete_ones.shape[0] > 0:
    au.scatter_aggregates(complete_ones, 'Stats for complete laps')
else:
    print('No complete laps yet.')
```




```
<Figure size 640x480 with 0 Axes>
```

2.6.4 Categories analysis

We're going back to comparing training results based on the training time, but in a different way. Instead of just scattering things in relation to iteration or episode number, this time we're grouping episodes based on a certaing information. For this we use function:

`analyze_categories(panda, category='quintile', groupcount=5, title=None)`

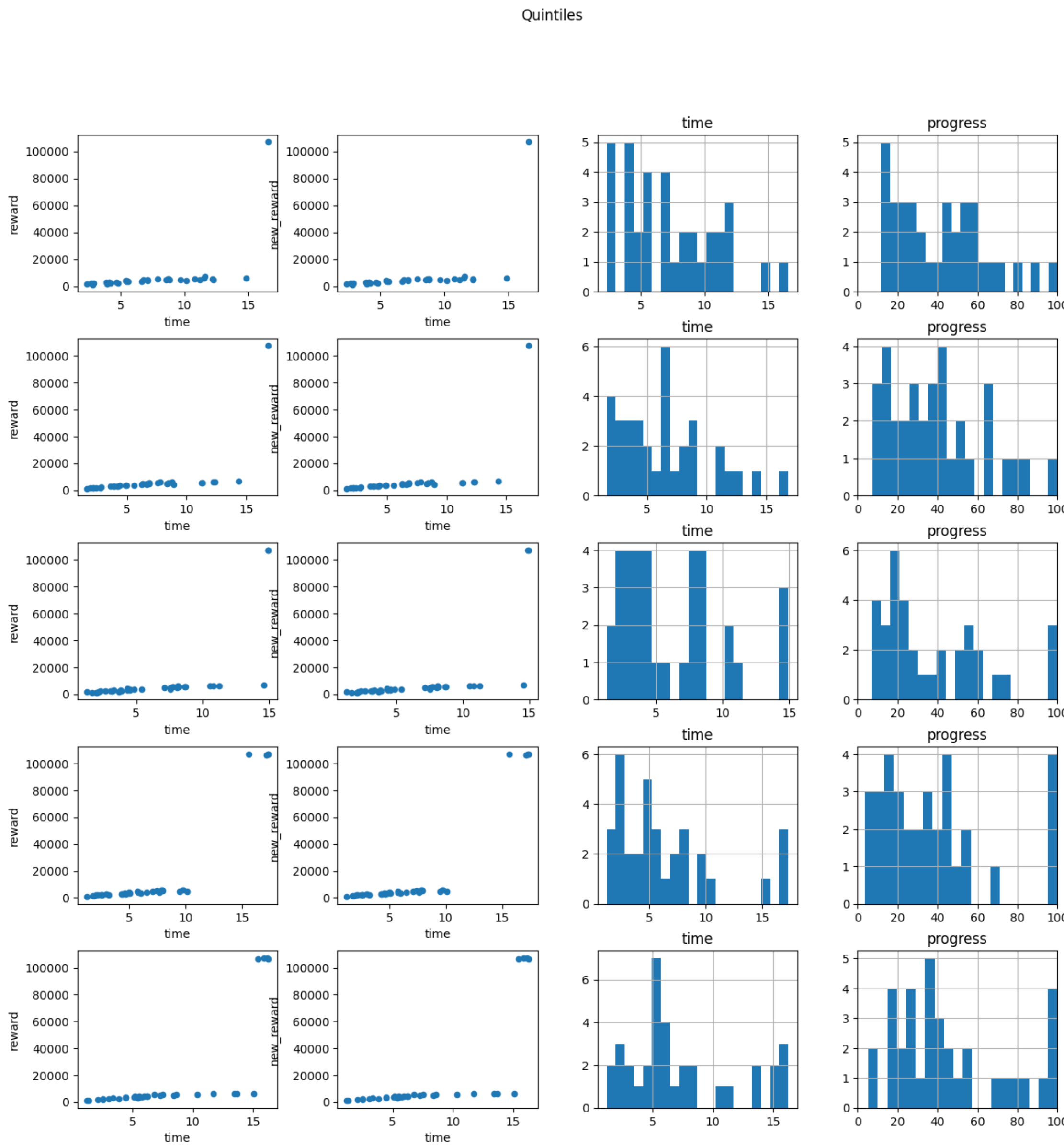
The idea is pretty simple - determine a way to cluster the data and provide that as the `category` parameter (alongside the count of groups available). In the default case we take advantage of the aggregated information to which quintile an episode belongs and thus build buckets each containing 20% of episodes which happened around the same time during the training. If your training lasted for five hours, this would show results grouped per each hour.

A side note: if you run the function with `category='start_at'` and `groupcount=20` you will get results based on the waypoint closest to the starting point of an episode. If you need to, you can introduce other types of categories and reuse the function.

The graphs are similar to what we've seen above. I especially like the progress one which shows where the model tends to struggle and whether it's successful laps rate is improving or beginning to decrease. Interestingly, I also had cases where I saw the completion drop on the progress rate only to improve in a later quintile, but with a better time graph.

A second side note: if you run this function for `complete_ones` instead of `simulation_agg`, suddenly the time histogram becomes more interesting as you can see whether completion times improve.

[11]:
```
au.scatter_by_groups(simulation_agg, title='Quintiles')
```




```
<Figure size 640x480 with 0 Axes>
```

2.7 Data in tables

While a lot can be seen in graphs that cannot be seen in the raw numbers, the numbers let us get into more detail. Below you will find a couple examples. If your model is behaving the way you would like it to, below tables may provide little added value, but if you struggle to improve your car's performance, they may come handy. In such cases I look for examples where high reward is giving to below-expected episode and when good episodes are given low reward.

You can then take the episode number and scatter it below, and also look at reward given per step - this can in turn draw your attention to some rewarding anomalies and help you detect some unexpected outcomes in your reward function.

There is a number of ways to select the data for display: * `nlargest`/`nsmallest` lets you display information based on a specific value being highest or lowest * filtering based on a field value, for instance `df[df['episode']==10]` will display only those steps in `df` which belong to episode 10 * `head()` lets you peek into a dataframe

There isn't a right set of tables to display here and the ones below may not suit your needs. Get to know Pandas more and have fun with them. It's almost as addictive as DeepRacer itself.

The examples have a short comment next to them explaining what they are showing.

[12]:
```python
# View ten best rewarded episodes in the training
simulation_agg.nlargest(10, 'new_reward')
```

[12]:
```
     iteration  episode  steps  start_at  progress    time       dist  \
97          10       97    225       132     100.0  14.924  23.688278
89           9       89    225        71     100.0  14.948  24.002670
157         16      157    243       132     100.0  16.131  23.323334
47           5       47    254        55     100.0  16.864  24.233769
164         17      164    239        32     100.0  15.874  22.996099
22           3       22    250        16     100.0  16.576  23.364256
135         14      135    261       117     100.0  17.335  22.303970
121         13      121    235         9     100.0  15.616  23.365259
155         16      155    232       117     100.0  15.407  22.524762
167         17      167    245        55     100.0  16.260  21.895888

      new_reward     speed       reward  time_if_complete  reward_if_complete  \
97   107511.4881  2.027600  107511.4881            14.924         107511.4881
89   107348.8902  2.022089  107348.8902            14.948         107348.8902
157  107317.8963  1.908848  107317.8963            16.131         107317.8963
47   107291.5678  1.911693  107291.5678            16.864         107291.5678
164  107232.7670  1.910251  107232.7670            15.874         107232.7670
22   107101.8551  1.878480  107101.8551            16.576         107101.8551
135  106957.6979  1.795556  106957.6979            17.335         106957.6979
121  106956.6303  1.875362  106956.6303            15.616         106956.6303
155  106878.1596  1.876767  106878.1596            15.407         106878.1596
167  106735.1456  1.826531  106735.1456            16.260         106735.1456

    quintile  complete
97       3rd         1
89       3rd         1
157      5th         1
47       2nd         1
164      5th         1
22       1st         1
135      4th         1
121      4th         1
155      5th         1
```

```
167       5th          1
```

[14]:
```python
# View five fastest complete laps
complete_ones.nsmallest(5, 'time')
```

[14]:
```
     iteration  episode  steps  start_at  progress    time       dist  \
97          10       97    225       132     100.0  14.924  23.688278
89           9       89    225        71     100.0  14.948  24.002670
155         16      155    232       117     100.0  15.407  22.524762
121         13      121    235         9     100.0  15.616  23.365259
164         17      164    239        32     100.0  15.874  22.996099

      new_reward     speed       reward  time_if_complete  reward_if_complete  \
97   107511.4881  2.027600  107511.4881            14.924         107511.4881
89   107348.8902  2.022089  107348.8902            14.948         107348.8902
155  106878.1596  1.876767  106878.1596            15.407         106878.1596
121  106956.6303  1.875362  106956.6303            15.616         106956.6303
164  107232.7670  1.910251  107232.7670            15.874         107232.7670

    quintile  complete
97       3rd         1
89       3rd         1
155      5th         1
121      4th         1
164      5th         1
```

[15]:
```python
# View five best rewarded completed laps
complete_ones.nlargest(5, 'reward')
```

[15]:
```
     iteration  episode  steps  start_at  progress    time       dist  \
97          10       97    225       132     100.0  14.924  23.688278
89           9       89    225        71     100.0  14.948  24.002670
157         16      157    243       132     100.0  16.131  23.323334
47           5       47    254        55     100.0  16.864  24.233769
164         17      164    239        32     100.0  15.874  22.996099

      new_reward     speed       reward  time_if_complete  reward_if_complete  \
97   107511.4881  2.027600  107511.4881            14.924         107511.4881
89   107348.8902  2.022089  107348.8902            14.948         107348.8902
157  107317.8963  1.908848  107317.8963            16.131         107317.8963
47   107291.5678  1.911693  107291.5678            16.864         107291.5678
164  107232.7670  1.910251  107232.7670            15.874         107232.7670

    quintile  complete
97       3rd         1
89       3rd         1
157      5th         1
```

```
47       2nd          1
164      5th          1
```

[16]:

```
# View five best rewarded in completed laps (according to new_reward if you are
↪using it)
complete_ones.nlargest(5, 'new_reward')
```

[16]:

```
     iteration  episode  steps  start_at  progress    time       dist  \
97          10       97    225       132     100.0  14.924  23.688278
89           9       89    225        71     100.0  14.948  24.002670
157         16      157    243       132     100.0  16.131  23.323334
47           5       47    254        55     100.0  16.864  24.233769
164         17      164    239        32     100.0  15.874  22.996099

      new_reward     speed       reward  time_if_complete  reward_if_complete  \
97   107511.4881  2.027600  107511.4881            14.924         107511.4881
89   107348.8902  2.022089  107348.8902            14.948         107348.8902
157  107317.8963  1.908848  107317.8963            16.131         107317.8963
47   107291.5678  1.911693  107291.5678            16.864         107291.5678
164  107232.7670  1.910251  107232.7670            15.874         107232.7670

    quintile  complete
97       3rd         1
89       3rd         1
157      5th         1
47       2nd         1
164      5th         1
```

[17]:

```
# View five most progressed episodes
simulation_agg.nlargest(5, 'progress')
```

[17]:

```
     iteration  episode  steps  start_at  progress    time       dist  \
22           3       22    250        16     100.0  16.576  23.364256
47           5       47    254        55     100.0  16.864  24.233769
89           9       89    225        71     100.0  14.948  24.002670
97          10       97    225       132     100.0  14.924  23.688278
121         13      121    235         9     100.0  15.616  23.365259

      new_reward     speed       reward  time_if_complete  reward_if_complete  \
22   107101.8551  1.878480  107101.8551            16.576         107101.8551
47   107291.5678  1.911693  107291.5678            16.864         107291.5678
89   107348.8902  2.022089  107348.8902            14.948         107348.8902
97   107511.4881  2.027600  107511.4881            14.924         107511.4881
121  106956.6303  1.875362  106956.6303            15.616         106956.6303

    quintile  complete
22       1st         1
```

```
47       2nd          1
89       3rd          1
97       3rd          1
121      4th          1
```

[18]:
```
# View information for a couple first episodes
simulation_agg.head()
```

[18]:
```
   iteration  episode  steps  start_at  progress    time       dist  \
0          1        0     60         1   15.8520   3.930   4.370350
1          1        1    131         9   57.3467   8.687  13.467511
2          1        2     43        16   11.9011   2.807   2.699413
3          1        3    185        24   64.0739  12.244  15.958547
4          1        4    163        32   59.0739  10.804  14.849314

   new_reward     speed     reward  time_if_complete  reward_if_complete  \
0   1857.9624  1.614333  1857.9624         24.791824        11720.681302
1   5716.7166  2.024122  5716.7166         15.148213         9968.693229
2   1332.0868  1.661628  1332.0868         23.586055        11192.972078
3   4990.4385  1.746378  4990.4385         19.109185         7788.566796
4   5690.6060  1.873313  5690.6060         18.288957         9633.029138

  quintile  complete
0      1st         0
1      1st         0
2      1st         0
3      1st         0
4      1st         0
```

[19]:
```
# Set maximum quantity of rows to view for a dataframe display - without that
# the view below will just hide some of the steps
pd.set_option('display.max_rows', 500)

# View all steps data for episode 10
df[df['episode']==10]
```

[19]:
```
      iteration  episode  steps       x       y      yaw  steering_angle  \
1088          2       10      1  8.2683  4.3957  75.1258          -15.00
1089          2       10      2  8.2683  4.3955  75.0983           11.72
1090          2       10      3  8.2725  4.4048  74.6866           28.79
1091          2       10      4  8.2748  4.4237  75.2334           13.82
1092          2       10      5  8.2732  4.4582  77.6261           -6.51
1093          2       10      6  8.2771  4.5081  80.0729          -15.00
1094          2       10      7  8.2874  4.5591  79.8311          -15.00
1095          2       10      8  8.3067  4.6165  78.5074           -6.43
1096          2       10      9  8.3248  4.6708  76.9119           12.05
1097          2       10     10  8.3406  4.7288  76.2682           15.03
```

1098	2	10	11	8.3459	4.7896	78.1035	-15.00
1099	2	10	12	8.3514	4.8632	80.5092	10.79
1100	2	10	13	8.3618	4.9468	81.2034	30.00
1101	2	10	14	8.3633	5.0308	84.0805	-6.20
1102	2	10	15	8.3580	5.1212	88.4375	30.00
1103	2	10	16	8.3539	5.2090	90.0409	4.84
1104	2	10	17	8.3343	5.3180	94.1315	30.00
1105	2	10	18	8.3108	5.4222	97.6081	-15.00
1106	2	10	19	8.2963	5.5076	98.3546	30.00
1107	2	10	20	8.2844	5.5974	98.2250	30.00
1108	2	10	21	8.2617	5.6742	100.8328	30.00
1109	2	10	22	8.2301	5.7504	104.5283	30.00
1110	2	10	23	8.1868	5.8162	110.7395	30.00
1111	2	10	24	8.1419	5.8676	116.2070	30.00
1112	2	10	25	8.0895	5.9196	121.7253	25.17
1113	2	10	26	8.0111	5.9745	130.1003	0.71
1114	2	10	27	7.9242	6.0319	136.7560	30.00
1115	2	10	28	7.8483	6.0819	141.2056	23.31
1116	2	10	29	7.7833	6.1225	143.2143	30.00
1117	2	10	30	7.7233	6.1548	145.5315	30.00
1118	2	10	31	7.6449	6.1858	149.7775	30.00
1119	2	10	32	7.5789	6.2022	154.1662	30.00
1120	2	10	33	7.4743	6.2073	163.0277	30.00
1121	2	10	34	7.3811	6.1973	172.3547	8.82
1122	2	10	35	7.2988	6.1815	-179.8102	2.91
1123	2	10	36	7.1921	6.1530	-172.1680	26.63
1124	2	10	37	7.0934	6.1162	-165.4875	-15.00
1125	2	10	38	6.9595	6.0579	-157.8526	-15.00
1126	2	10	39	6.8608	6.0286	-156.2856	-15.00
1127	2	10	40	6.7377	6.0074	-160.8949	-15.00
1128	2	10	41	6.6469	5.9998	-165.7057	-15.00
1129	2	10	42	6.5448	5.9991	-171.0369	-0.97
1130	2	10	43	6.4474	6.0018	-175.0834	30.00
1131	2	10	44	6.3190	5.9970	-176.1235	-15.00
1132	2	10	45	6.2528	5.9825	-173.1747	14.18
1133	2	10	46	6.1620	5.9641	-171.2493	-15.00
1134	2	10	47	6.0720	5.9535	-172.0181	-0.24
1135	2	10	48	6.0290	5.9532	-173.3271	17.39
1136	2	10	49	5.9506	5.9451	-173.8147	30.00
1137	2	10	50	5.8698	5.9252	-172.0166	-6.06
1138	2	10	51	5.8065	5.9072	-169.9745	-10.10
1139	2	10	52	5.7165	5.8989	-171.5923	30.00
1140	2	10	53	5.6349	5.8972	-175.3139	-1.27
1141	2	10	54	5.5657	5.8920	-176.8008	7.66
1142	2	10	55	5.4948	5.8820	-175.5118	30.00
1143	2	10	56	5.4093	5.8579	-171.6441	-15.00
1144	2	10	57	5.3312	5.8271	-167.3315	30.00

1145	2	10	58	5.2354	5.7994	-166.0767	-15.00
1146	2	10	59	5.1492	5.7761	-166.1177	30.00
1147	2	10	60	5.0515	5.7500	-165.9555	30.00
1148	2	10	61	4.9546	5.7037	-161.3185	25.36
1149	2	10	62	4.8558	5.6332	-152.4449	21.58
1150	2	10	63	4.7791	5.5703	-147.7196	20.38
1151	2	10	64	4.7036	5.4931	-142.1962	3.50
1152	2	10	65	4.6479	5.4332	-139.1218	30.00
1153	2	10	66	4.5906	5.3687	-136.5600	30.00
1154	2	10	67	4.5303	5.2796	-131.4313	30.00
1155	2	10	68	4.4967	5.1994	-124.1017	18.44
1156	2	10	69	4.4645	5.0990	-116.8033	18.06
1157	2	10	70	4.4455	5.0058	-111.1337	30.00
1158	2	10	71	4.4358	4.9176	-106.0680	-15.00
1159	2	10	72	4.4270	4.8375	-103.0436	-15.00
1160	2	10	73	4.3973	4.7490	-105.0209	-15.00
1161	2	10	74	4.3645	4.6818	-108.2758	-13.47
1162	2	10	75	4.3128	4.5917	-112.8230	-15.00
1163	2	10	76	4.2503	4.5116	-118.8614	9.77
1164	2	10	77	4.1889	4.4343	-123.3722	24.15
1165	2	10	78	4.1216	4.3341	-123.6681	-8.72
1166	2	10	79	4.0531	4.2268	-123.1130	-15.00
1167	2	10	80	3.9767	4.1232	-124.1293	-15.00
1168	2	10	81	3.9066	4.0509	-127.9479	-15.00
1169	2	10	82	3.8250	3.9793	-132.5164	-15.00
1170	2	10	83	3.7174	3.9098	-139.2573	-15.00
1171	2	10	84	3.6180	3.8652	-146.9787	-15.00
1172	2	10	85	3.4987	3.8231	-155.5843	-15.00
1173	2	10	86	3.3815	3.8052	-167.0679	-15.00
1174	2	10	87	3.2876	3.8005	-176.5932	-10.66
1175	2	10	88	3.1983	3.8148	170.1309	-10.50
1176	2	10	89	3.1096	3.8372	157.8734	2.95
1177	2	10	90	3.0270	3.8760	144.7335	11.99
1178	2	10	91	2.9476	3.9245	134.4852	23.93
1179	2	10	92	2.8535	3.9990	122.7180	12.56
1180	2	10	93	2.7793	4.0676	114.8140	30.00
1181	2	10	94	2.7018	4.1479	108.8846	30.00
1182	2	10	95	2.6292	4.2294	105.2225	30.00
1183	2	10	96	2.5725	4.2967	103.4024	20.58
1184	2	10	97	2.5194	4.3664	104.6602	8.23
1185	2	10	98	2.4654	4.4611	105.5810	0.25
1186	2	10	99	2.4288	4.5488	106.3878	9.42
1187	2	10	100	2.3879	4.6378	109.4060	10.11
1188	2	10	101	2.3246	4.7447	114.5953	30.00
1189	2	10	102	2.2635	4.8220	120.2029	-0.01
1190	2	10	103	2.1734	4.9284	126.9188	-0.09

	speed	action	reward	done	on_track	progress	closest_waypoint	\
1088	3.03	-1	0.0000	0	True	0.6062	78	
1089	3.59	-1	100.7954	0	True	0.6054	78	
1090	4.00	-1	104.6000	0	True	0.6488	79	
1091	2.82	-1	87.7826	0	True	0.7304	79	
1092	0.84	-1	0.0010	0	True	0.8745	79	
1093	4.00	-1	104.2000	0	True	1.0912	79	
1094	0.60	-1	0.0010	0	True	1.3161	80	
1095	0.60	-1	0.0010	0	True	1.5756	80	
1096	3.73	-1	101.3539	0	True	1.8196	80	
1097	3.44	-1	97.8001	0	True	2.0777	81	
1098	2.24	-1	71.9720	0	True	2.3386	81	
1099	1.24	-1	0.0010	0	True	2.6578	82	
1100	4.00	-1	103.2667	0	True	3.0169	82	
1101	0.60	-1	0.0010	0	True	3.3801	83	
1102	2.29	-1	72.8486	0	True	3.7656	83	
1103	3.52	-1	98.0404	0	True	4.1458	84	
1104	0.60	-1	0.0010	0	True	4.6158	85	
1105	0.60	-1	0.0010	0	True	5.0646	85	
1106	1.25	-1	0.0010	0	True	5.4326	86	
1107	0.66	-1	0.0010	0	True	5.7624	86	
1108	2.72	-1	83.1765	0	True	6.0974	87	
1109	0.60	-1	0.0010	0	True	6.3584	87	
1110	0.60	-1	0.0010	0	True	6.6949	88	
1111	4.00	-1	101.8000	0	True	6.8959	88	
1112	4.00	-1	101.6667	0	True	7.2133	89	
1113	4.00	-1	101.5333	0	True	7.5516	89	
1114	0.60	-1	0.0010	0	True	8.0015	90	
1115	0.60	-1	0.0010	0	True	8.3212	90	
1116	0.60	-1	0.0010	0	True	8.6495	91	
1117	3.22	-1	91.9383	0	True	8.8812	91	
1118	4.00	-1	100.8667	0	True	9.2457	92	
1119	2.29	-1	70.5734	0	True	9.4871	92	
1120	0.60	-1	0.0010	0	True	9.9296	93	
1121	2.19	-1	67.6229	0	True	10.2762	93	
1122	2.55	-1	77.0284	0	True	10.6115	94	
1123	3.22	-1	91.1800	0	True	11.0315	94	
1124	4.00	-1	100.0667	0	True	11.4341	95	
1125	0.60	-1	0.0010	0	True	11.9870	96	
1126	0.60	-1	0.0010	0	True	12.4209	97	
1127	0.60	-1	0.0010	0	True	12.9735	97	
1128	2.48	-1	74.5030	0	True	13.3656	98	
1129	0.60	-1	0.0010	0	True	13.8323	99	
1130	3.69	-1	96.4151	0	True	14.2756	99	
1131	0.60	-1	0.0010	0	True	14.8531	100	
1132	0.60	-1	0.0010	0	True	15.1438	101	
1133	0.60	-1	0.0010	0	True	15.5721	101	

1134	2.85	−1	82.5364	0	True	15.9638	102
1135	4.00	−1	98.6000	0	True	16.1900	102
1136	0.60	−1	0.0010	0	True	16.5289	103
1137	2.64	−1	77.2608	0	True	16.9190	103
1138	4.00	−1	98.2000	0	True	17.2032	104
1139	0.60	−1	0.0010	0	True	17.6129	104
1140	0.60	−1	0.0010	0	True	17.9726	105
1141	1.13	−1	0.0010	0	True	18.2627	105
1142	1.88	−1	56.5719	0	True	18.5661	106
1143	2.11	−1	62.5349	0	True	18.9642	107
1144	1.16	−1	0.0010	0	True	19.3470	107
1145	4.00	−1	97.2667	0	True	19.7701	108
1146	3.23	−1	88.1686	0	True	20.1567	108
1147	3.84	−1	95.6325	0	True	20.5681	109
1148	0.94	−1	0.0010	0	True	21.0030	110
1149	1.62	−1	49.8481	0	True	21.4980	110
1150	0.88	−1	0.0010	0	True	21.8889	111
1151	0.60	−1	0.0010	0	True	22.3530	112
1152	1.79	−1	53.0759	0	True	22.6611	112
1153	3.89	−1	95.2593	0	True	23.0290	113
1154	0.60	−1	0.0010	0	True	23.4470	113
1155	1.77	−1	52.2105	0	True	23.8216	114
1156	0.60	−1	0.0010	0	True	24.2386	115
1157	1.03	−1	0.0010	0	True	24.6216	115
1158	1.68	−1	49.9641	0	True	24.9936	116
1159	0.60	−1	0.0010	0	True	25.3265	116
1160	1.94	−1	55.6418	0	True	25.7300	117
1161	4.00	−1	95.1333	0	True	26.0543	117
1162	3.88	−1	94.0137	0	True	26.5154	118
1163	1.05	−1	0.0010	0	True	26.9401	119
1164	4.00	−1	94.7333	0	True	27.4095	119
1165	3.62	−1	91.0327	0	True	28.0210	120
1166	0.60	−1	0.0010	0	True	28.6638	121
1167	0.65	−1	0.0010	0	True	29.2688	122
1168	4.00	−1	94.2000	0	True	29.6845	123
1169	2.57	−1	71.4455	0	True	30.1460	124
1170	0.60	−1	0.0010	0	True	30.6759	124
1171	2.30	−1	63.8035	0	True	31.1297	125
1172	0.60	−1	0.0010	0	True	31.6731	126
1173	3.94	−1	93.0413	0	True	32.1729	127
1174	0.60	−1	0.0010	0	True	32.5607	127
1175	4.00	−1	93.2667	0	True	32.9381	128
1176	2.26	−1	62.2536	0	True	33.3080	128
1177	4.00	−1	93.0000	0	True	33.6772	129
1178	2.19	−1	60.0989	0	True	34.0370	130
1179	1.35	−1	0.0010	0	True	34.5418	130
1180	2.92	−1	78.0516	0	True	35.0173	131

1181	0.60	-1	0.0010	0	True	35.4463	132
1182	0.94	-1	0.0010	0	True	35.9834	133
1183	3.96	-1	91.8497	0	True	36.4004	133
1184	1.89	-1	51.3732	0	True	36.7433	134
1185	0.60	-1	0.0010	0	True	37.1259	134
1186	1.68	-1	46.2521	0	True	37.4395	135
1187	4.00	-1	91.6667	0	False	37.6248	135
1188	4.00	-1	91.5333	0	False	38.0599	136
1189	2.01	-1	53.7317	0	False	38.2279	136
1190	1.11	-1	0.0010	1	False	38.6612	137

	track_len	tstamp	episode_status	pause_duration	new_reward
1088	23.12	160.52	prepare	0.0	0.0000
1089	23.12	160.583	in_progress	0.0	100.7954
1090	23.12	160.652	in_progress	0.0	104.6000
1091	23.12	160.723	in_progress	0.0	87.7826
1092	23.12	160.788	in_progress	0.0	0.0010
1093	23.12	160.857	in_progress	0.0	104.2000
1094	23.12	160.921	in_progress	0.0	0.0010
1095	23.12	160.953	in_progress	0.0	0.0010
1096	23.12	161.058	in_progress	0.0	101.3539
1097	23.12	161.123	in_progress	0.0	97.8001
1098	23.12	161.191	in_progress	0.0	71.9720
1099	23.12	161.253	in_progress	0.0	0.0010
1100	23.12	161.319	in_progress	0.0	103.2667
1101	23.12	161.388	in_progress	0.0	0.0010
1102	23.12	161.454	in_progress	0.0	72.8486
1103	23.12	161.524	in_progress	0.0	98.0404
1104	23.12	161.582	in_progress	0.0	0.0010
1105	23.12	161.656	in_progress	0.0	0.0010
1106	23.12	161.721	in_progress	0.0	0.0010
1107	23.12	161.789	in_progress	0.0	0.0010
1108	23.12	161.851	in_progress	0.0	83.1765
1109	23.12	161.918	in_progress	0.0	0.0010
1110	23.12	161.979	in_progress	0.0	0.0010
1111	23.12	162.056	in_progress	0.0	101.8000
1112	23.12	162.12	in_progress	0.0	101.6667
1113	23.12	162.166	in_progress	0.0	101.5333
1114	23.12	162.253	in_progress	0.0	0.0010
1115	23.12	162.322	in_progress	0.0	0.0010
1116	23.12	162.386	in_progress	0.0	0.0010
1117	23.12	162.449	in_progress	0.0	91.9383
1118	23.12	162.511	in_progress	0.0	100.8667
1119	23.12	162.587	in_progress	0.0	70.5734
1120	23.12	162.637	in_progress	0.0	0.0010
1121	23.12	162.719	in_progress	0.0	67.6229
1122	23.12	162.785	in_progress	0.0	77.0284

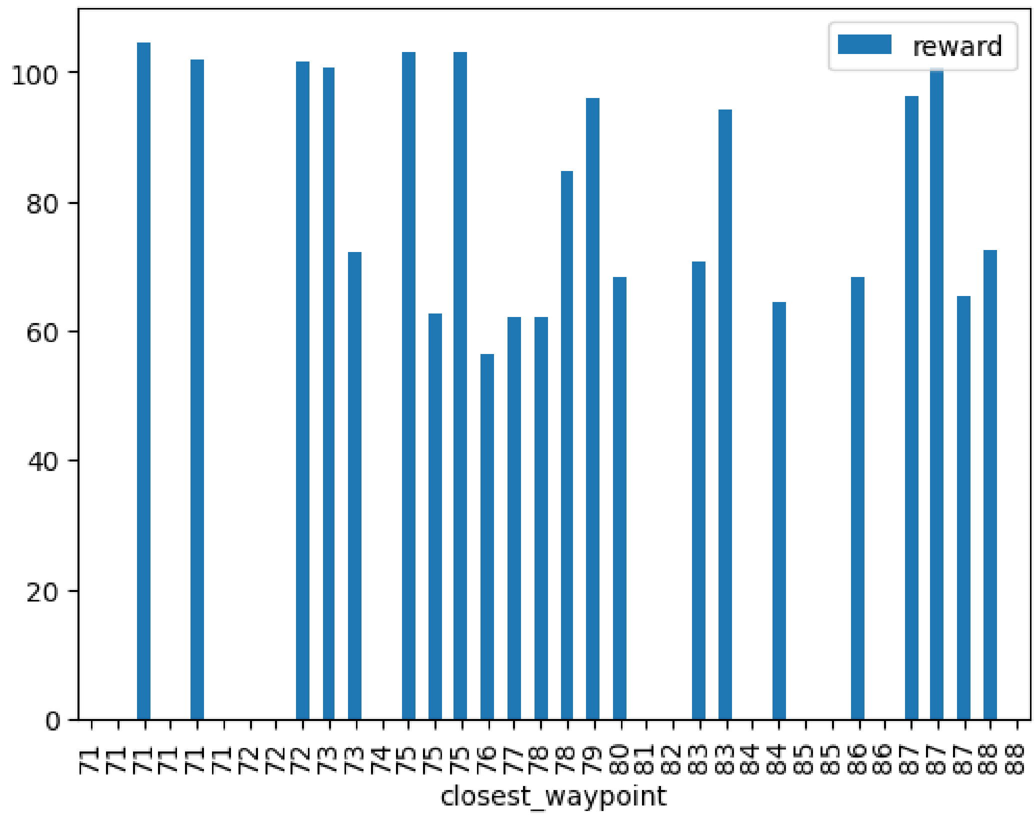
1123	23.12	162.85	in_progress	0.0	91.1800
1124	23.12	162.923	in_progress	0.0	100.0667
1125	23.12	162.979	in_progress	0.0	0.0010
1126	23.12	163.059	in_progress	0.0	0.0010
1127	23.12	163.119	in_progress	0.0	0.0010
1128	23.12	163.185	in_progress	0.0	74.5030
1129	23.12	163.252	in_progress	0.0	0.0010
1130	23.12	163.314	in_progress	0.0	96.4151
1131	23.12	163.385	in_progress	0.0	0.0010
1132	23.12	163.451	in_progress	0.0	0.0010
1133	23.12	163.519	in_progress	0.0	0.0010
1134	23.12	163.585	in_progress	0.0	82.5364
1135	23.12	163.645	in_progress	0.0	98.6000
1136	23.12	163.721	in_progress	0.0	0.0010
1137	23.12	163.788	in_progress	0.0	77.2608
1138	23.12	163.853	in_progress	0.0	98.2000
1139	23.12	163.923	in_progress	0.0	0.0010
1140	23.12	163.986	in_progress	0.0	0.0010
1141	23.12	164.049	in_progress	0.0	0.0010
1142	23.12	164.127	in_progress	0.0	56.5719
1143	23.12	164.186	in_progress	0.0	62.5349
1144	23.12	164.243	in_progress	0.0	0.0010
1145	23.12	164.281	in_progress	0.0	97.2667
1146	23.12	164.389	in_progress	0.0	88.1686
1147	23.12	164.457	in_progress	0.0	95.6325
1148	23.12	164.522	in_progress	0.0	0.0010
1149	23.12	164.572	in_progress	0.0	49.8481
1150	23.12	164.659	in_progress	0.0	0.0010
1151	23.12	164.72	in_progress	0.0	0.0010
1152	23.12	164.764	in_progress	0.0	53.0759
1153	23.12	164.86	in_progress	0.0	95.2593
1154	23.12	164.915	in_progress	0.0	0.0010
1155	23.12	164.985	in_progress	0.0	52.2105
1156	23.12	165.051	in_progress	0.0	0.0010
1157	23.12	165.121	in_progress	0.0	0.0010
1158	23.12	165.185	in_progress	0.0	49.9641
1159	23.12	165.257	in_progress	0.0	0.0010
1160	23.12	165.321	in_progress	0.0	55.6418
1161	23.12	165.388	in_progress	0.0	95.1333
1162	23.12	165.457	in_progress	0.0	94.0137
1163	23.12	165.498	in_progress	0.0	0.0010
1164	23.12	165.538	in_progress	0.0	94.7333
1165	23.12	165.653	in_progress	0.0	91.0327
1166	23.12	165.721	in_progress	0.0	0.0010
1167	23.12	165.79	in_progress	0.0	0.0010
1168	23.12	165.858	in_progress	0.0	94.2000
1169	23.12	165.917	in_progress	0.0	71.4455

```
1170      23.12  165.989    in_progress            0.0      0.0010
1171      23.12  166.066    in_progress            0.0     63.8035
1172      23.12   166.12    in_progress            0.0      0.0010
1173      23.12  166.183    in_progress            0.0     93.0413
1174      23.12  166.248    in_progress            0.0      0.0010
1175      23.12  166.319    in_progress            0.0     93.2667
1176      23.12   166.39    in_progress            0.0     62.2536
1177      23.12  166.454    in_progress            0.0     93.0000
1178      23.12  166.521    in_progress            0.0     60.0989
1179      23.12  166.549    in_progress            0.0      0.0010
1180      23.12  166.658    in_progress            0.0     78.0516
1181      23.12   166.72    in_progress            0.0      0.0010
1182      23.12  166.765    in_progress            0.0      0.0010
1183      23.12  166.856    in_progress            0.0     91.8497
1184      23.12  166.915    in_progress            0.0     51.3732
1185      23.12  166.986    in_progress            0.0      0.0010
1186      23.12  167.054    in_progress            0.0     46.2521
1187      23.12  167.121    in_progress            0.0     91.6667
1188      23.12  167.189    in_progress            0.0     91.5333
1189      23.12  167.254    in_progress            0.0     53.7317
1190      23.12  167.327      off_track            0.0      0.0010
```

2.8 Analyze the reward distribution for your reward function

[20]:

```
# This shows a histogram of actions per closest waypoint for episode 889.
# Will let you spot potentially problematic places in reward granting.
# In this example reward function is clearly `return 1`. It may be worrying
# if your reward function has some logic in it.
# If you have a final step reward that makes the rest of this histogram
# unreadable, you can filter the last step out by using
# `episode[:-1].plot.bar` instead of `episode.plot.bar`
episode = df[df['episode']==9]

if episode.empty:
    print("You probably don't have episode with this number, try a lower one.")
else:
    episode.plot.bar(x='closest_waypoint', y='reward')
```



2.8.1 Path taken for top reward iterations

NOTE: at some point in the past in a single episode the car could go around multiple laps, the episode was terminated when car completed 1000 steps. Currently one episode has at most one lap. This explains why you can see multiple laps in an episode plotted below.

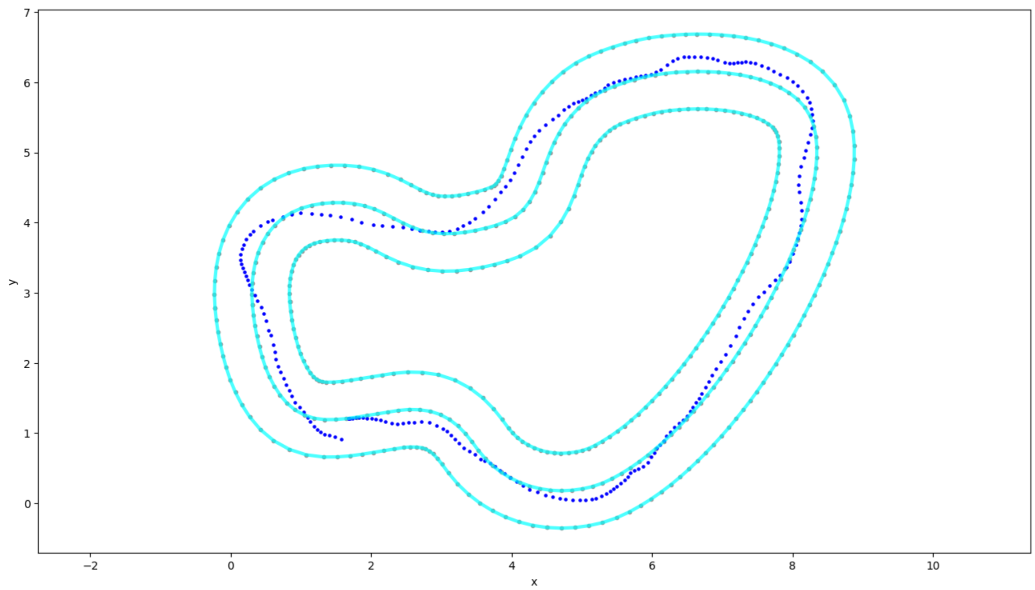
Being able to plot the car's route in an episode can help you detect certain patterns in its behaviours and either promote them more or train away from them. While being able to watch the car go in the training gives some information, being able to reproduce it after the training is much more practical.

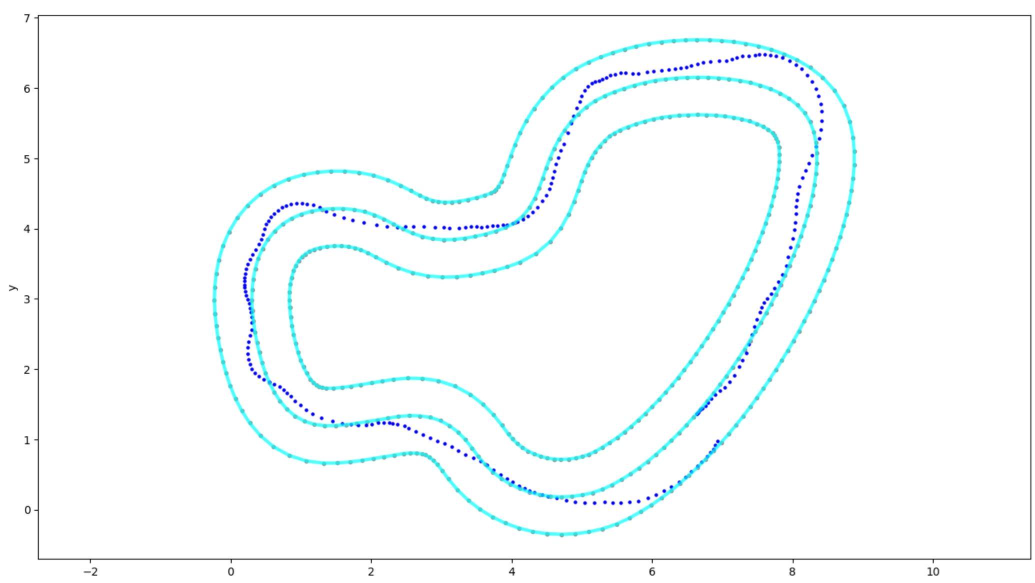
Graphs below give you a chance to look deeper into your car's behaviour on track.

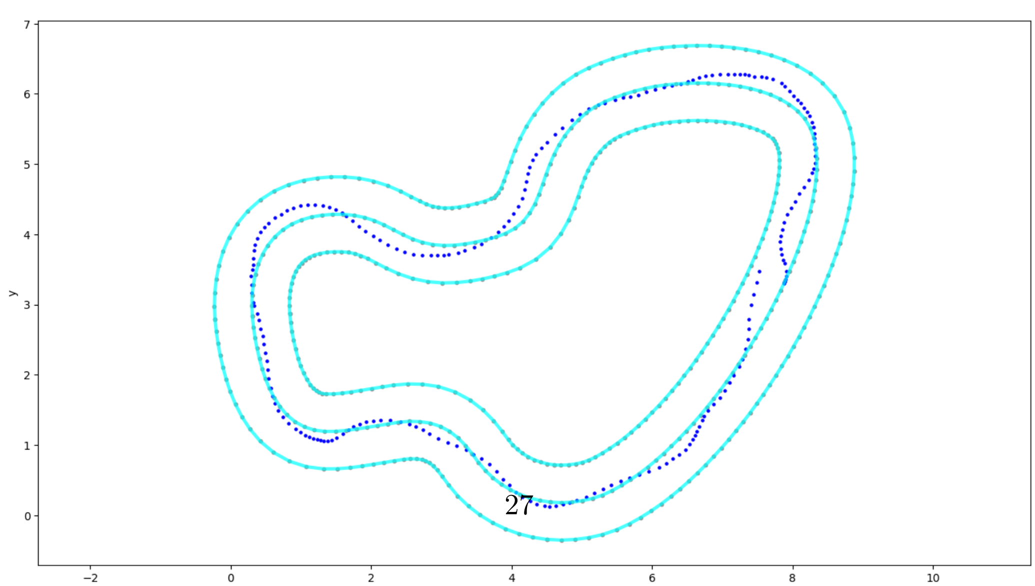
We start with plot_selected_laps. The general idea of this block is as follows: * Select laps(episodes) that have the properties that you care about, for instance, fastest, most progressed, failing in a certain section of the track or not failing in there, * Provide the list of them in a dataframe into the plot_selected_laps, together with the whole training dataframe and the track info, * You've got the laps to analyse.

[21]:
```
# Some examples:
# highest reward for complete laps:
# episodes_to_plot = complete_ones.nlargest(3,'reward')
```

```python
# highest progress from all episodes:
episodes_to_plot = simulation_agg.nlargest(3,'progress')

pu.plot_selected_laps(episodes_to_plot, df, track)
```


y
x


y
x


y
x

```
&lt;Figure size 640x480 with 0 Axes&gt;
```

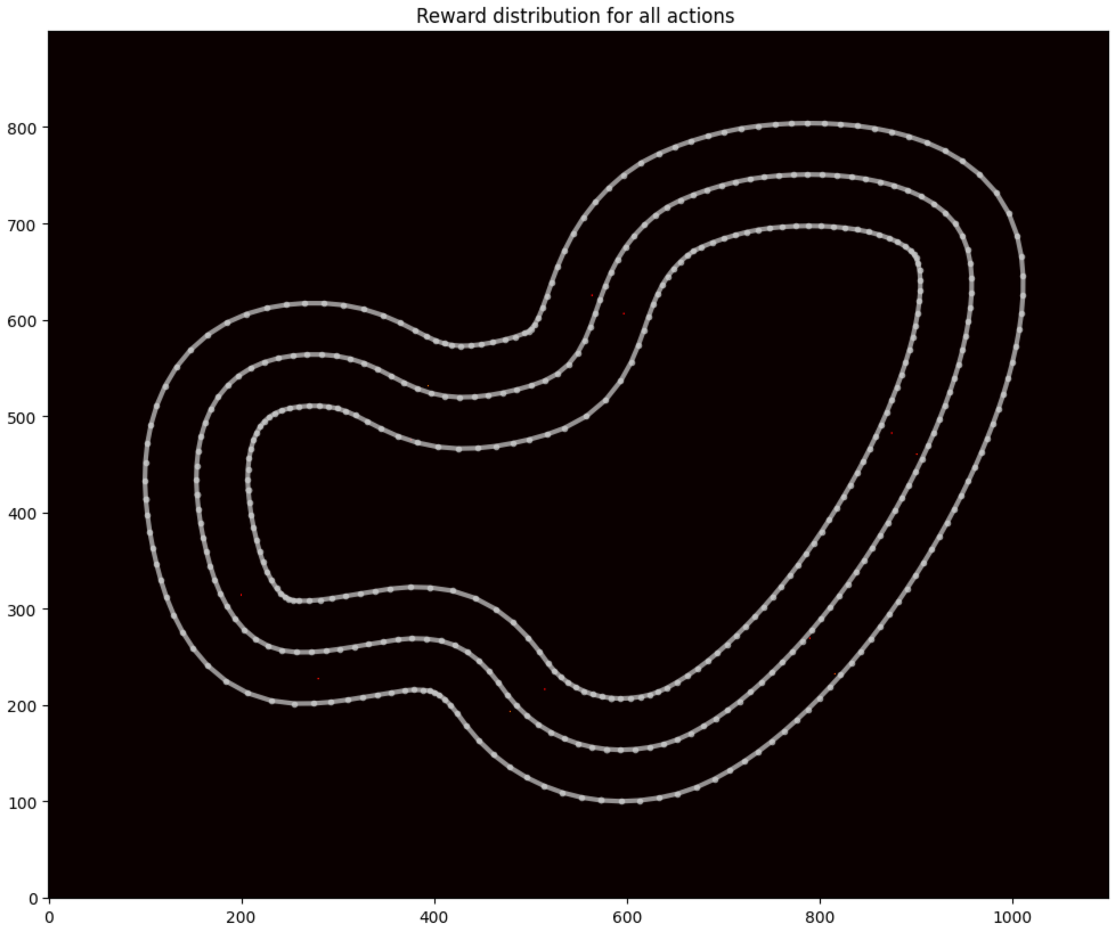
2.8.2 Plot a heatmap of rewards for current training.

The brighter the colour, the higher the reward granted in given coordinates. If instead of a similar view as in the example below you get a dark image with hardly any dots, it might be that your rewards are highly disproportionate and possibly sparse.

Disproportion means you may have one reward of 10.000 and the rest in range 0.01-1. In such cases the vast majority of dots will simply be very dark and the only bright dot might be in a place difficult to spot. I recommend you go back to the tables and show highest and average rewards per step to confirm if this is the case. Such disproportions may not affect your traning very negatively, but they will make the data less readable in this notebook.

Sparse data means that the car gets a high reward for the best behaviour and very low reward for anything else, and worse even, reward is pretty much discrete (return 10 for narrow perfect, else return 0.1). The car relies on reward varying between behaviours to find gradients that can lead to improvement. If that is missing, the model will struggle to improve.

[22]:
```
#If you'd like some other colour criterion, you can add
#a value_field parameter and specify a different column

pu.plot_track(df, track)
```

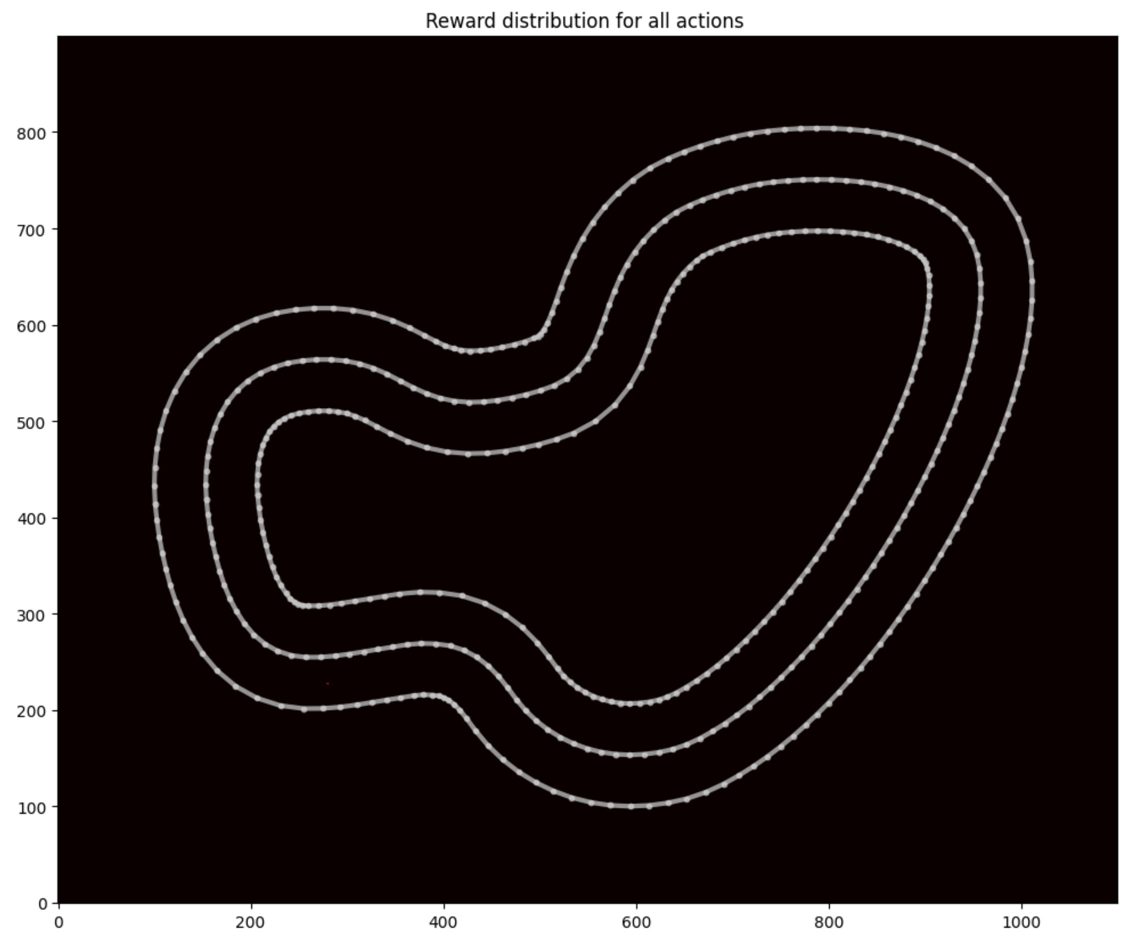



```
<Figure size 640x480 with 0 Axes>
```

2.8.3 Plot a particular iteration

This is same as the heatmap above, but just for a single iteration.

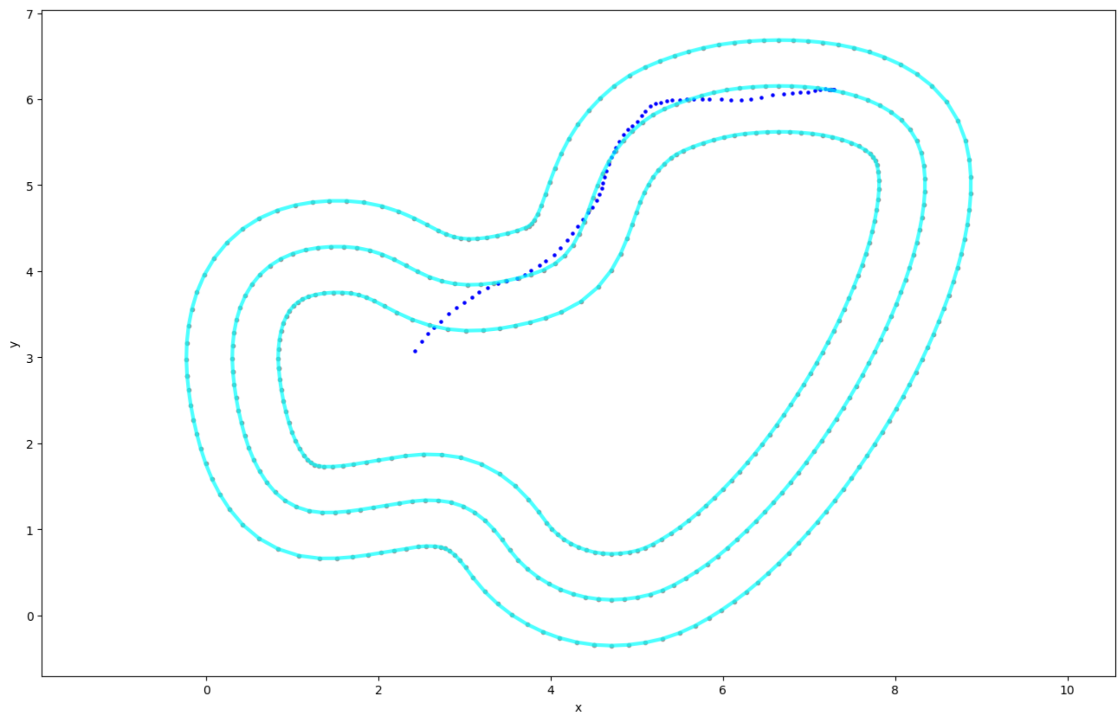
[23]:
```
#If you'd like some other colour criterion, you can add
#a value_field parameter and specify a different column
iteration_id = 3

pu.plot_track(df[df['iteration'] == iteration_id], track)
```



<Figure size 640x480 with 0 Axes>

2.8.4 Path taken in a particular episode

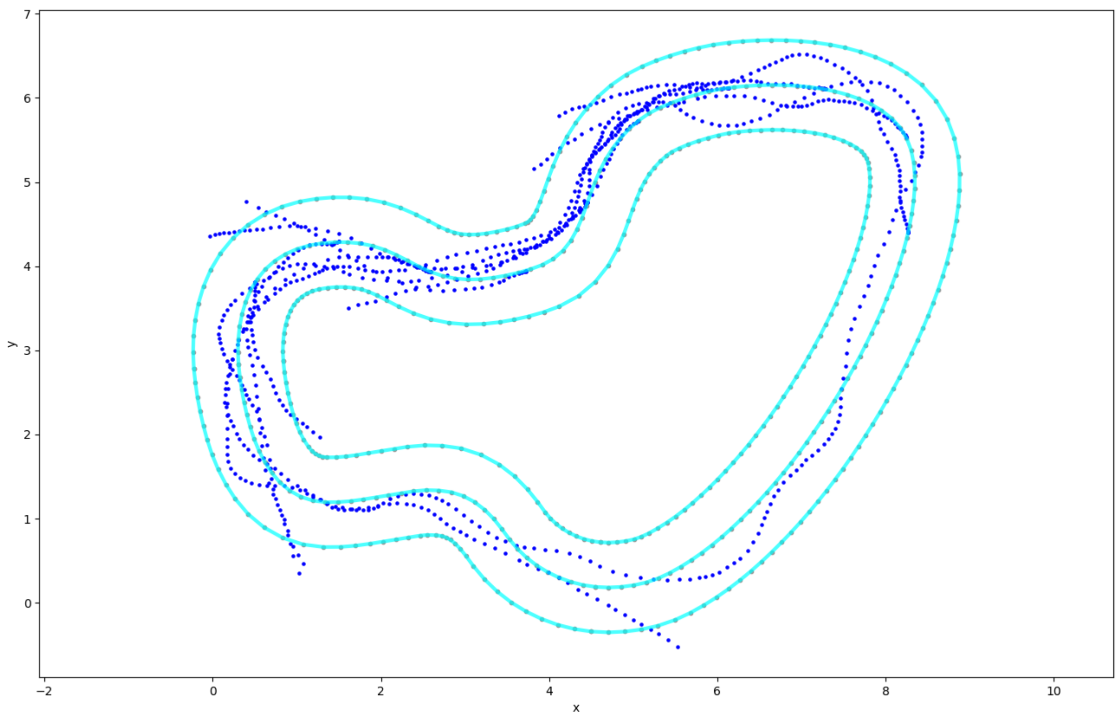
[24]:
```
episode_id = 12

pu.plot_selected_laps([episode_id], df, track)
```



<Figure size 640x480 with 0 Axes>

2.8.5 Path taken in a particular iteration

[25]:
```
iteration_id = 10

pu.plot_selected_laps([iteration_id], df, track, section_to_plot = 'iteration')
```



<Figure size 640x480 with 0 Axes>

3 Action breakdown per iteration and historgram for action distribution for each of the turns - reinvent track

This plot is useful to understand the actions that the model takes for any given iteration. Unfortunately at this time it is not fit for purpose as it assumes six actions in the action space and has other issues. It will require some work to get it to done but the information it returns will be very valuable.

This is a bit of an attempt to abstract away from the brilliant function in the original notebook towards a more general graph that we could use. It should be treated as a work in progress. The track_breakdown could be used as a starting point for a general track information object to handle all the customisations needed in methods of this notebook.

A breakdown track data needs to be available for it. If you cannot find it for the desired track, MAKEIT.

Currently supported tracks:

[26]:
```
track_breakdown.keys()
```

[26]:
```
dict_keys(['reinvent2018', 'london_loop'])
```

You can replace episode_ids with iteration_ids and make a breakdown for a whole iteration.

Note: does not work for continuous action space (yet).

```
[ ]: abu.action_breakdown(df, track, track_breakdown=track_breakdown.
      ↪get('reinvent2018'), episode_ids=[12])
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```