Training_analysis

October 31, 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
```

```
Requirement already satisfied: deepracer-utils in
/opt/conda/lib/python3.10/site-packages (1.0.1)
Requirement already satisfied: numpy>=1.18.0 in /opt/conda/lib/python3.10/site-
packages (from deepracer-utils) (1.23.4)
Requirement already satisfied: matplotlib>=3.1.0 in
/opt/conda/lib/python3.10/site-packages (from deepracer-utils) (3.6.1)
Requirement already satisfied: boto3>=1.12.0 in /opt/conda/lib/python3.10/site-
packages (from deepracer-utils) (1.24.91)
Requirement already satisfied: joblib>=0.17.0 in /opt/conda/lib/python3.10/site-
packages (from deepracer-utils) (1.2.0)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in
/opt/conda/lib/python3.10/site-packages (from deepracer-utils) (2.8.2)
Requirement already satisfied: shapely>=1.7.0 in /opt/conda/lib/python3.10/site-
packages (from deepracer-utils) (1.8.5.post1)
Requirement already satisfied: scikit-learn>=0.22.0 in
/opt/conda/lib/python3.10/site-packages (from deepracer-utils) (1.1.2)
Requirement already satisfied: pandas>=1.0.0 in /opt/conda/lib/python3.10/site-
packages (from deepracer-utils) (1.5.0)
Requirement already satisfied: botocore<1.28.0,>=1.27.91 in
/opt/conda/lib/python3.10/site-packages (from boto3>=1.12.0->deepracer-utils)
(1.27.91)
Requirement already satisfied: s3transfer<0.7.0,>=0.6.0 in
/opt/conda/lib/python3.10/site-packages (from boto3>=1.12.0->deepracer-utils)
(0.6.0)
```

```
Requirement already satisfied: jmespath<2.0.0,>=0.7.1 in
/opt/conda/lib/python3.10/site-packages (from boto3>=1.12.0->deepracer-utils)
(1.0.1)
Requirement already satisfied: pillow>=6.2.0 in /opt/conda/lib/python3.10/site-
packages (from matplotlib>=3.1.0->deepracer-utils) (9.2.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.10/site-
packages (from matplotlib>=3.1.0->deepracer-utils) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.10/site-packages (from matplotlib>=3.1.0->deepracer-
utils) (4.37.4)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib>=3.1.0->deepracer-
utils) (1.4.4)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib>=3.1.0->deepracer-
utils) (1.0.5)
Requirement already satisfied: pyparsing>=2.2.1 in
/opt/conda/lib/python3.10/site-packages (from matplotlib>=3.1.0->deepracer-
utils) (3.0.9)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.10/site-packages (from matplotlib>=3.1.0->deepracer-
utils) (21.3)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.10/site-
packages (from pandas>=1.0.0->deepracer-utils) (2022.4)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-
packages (from python-dateutil<3.0.0,>=2.1->deepracer-utils) (1.16.0)
Requirement already satisfied: scipy>=1.3.2 in /opt/conda/lib/python3.10/site-
packages (from scikit-learn>=0.22.0->deepracer-utils) (1.9.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/conda/lib/python3.10/site-packages (from scikit-learn>=0.22.0->deepracer-
utils) (3.1.0)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in
/opt/conda/lib/python3.10/site-packages (from
botocore<1.28.0,>=1.27.91->boto3>=1.12.0->deepracer-utils) (1.26.11)
```

2.3 Imports

Run the imports block below:

[2]:
```
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, \
```

```python
    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.

[4]:
```python
model_logs_root = 'compressed_logs/Practice-2-latest'
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': 0.008,
 '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': 8,
 'num_epochs': 5,
 '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:

[5]:
```
df.head()
```

[5]:
```
   iteration  episode  steps       x       y      yaw  steering_angle  speed  \
0          1        0      3  0.3283  2.6840 -82.8478            2.31   0.60
1          1        0      4  0.3335  2.6639 -81.9414           30.00   3.51
2          1        0      5  0.3422  2.6352 -80.8169           30.00   3.14
3          1        0      6  0.3693  2.5936 -76.0889           13.73   4.00
4          1        0      7  0.3996  2.5500 -71.6002           16.38   0.60

   action    reward  done on_track  progress  closest_waypoint  track_len  \
0      -1   40.0010     0     True    0.6399                 1      23.12
1      -1  139.5474     0     True    0.7301                 1      23.12
2      -1  133.9503     0     True    0.8583                 1      23.12
3      -1  144.2000     0     True    1.0527                 2      23.12
4      -1   40.0010     0     True    1.2582                 2      23.12

   tstamp episode_status  pause_duration
0  22.803    in_progress             0.0
1  22.863    in_progress             0.0
2  22.929    in_progress             0.0
3  22.999    in_progress             0.0
4  23.064    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
```

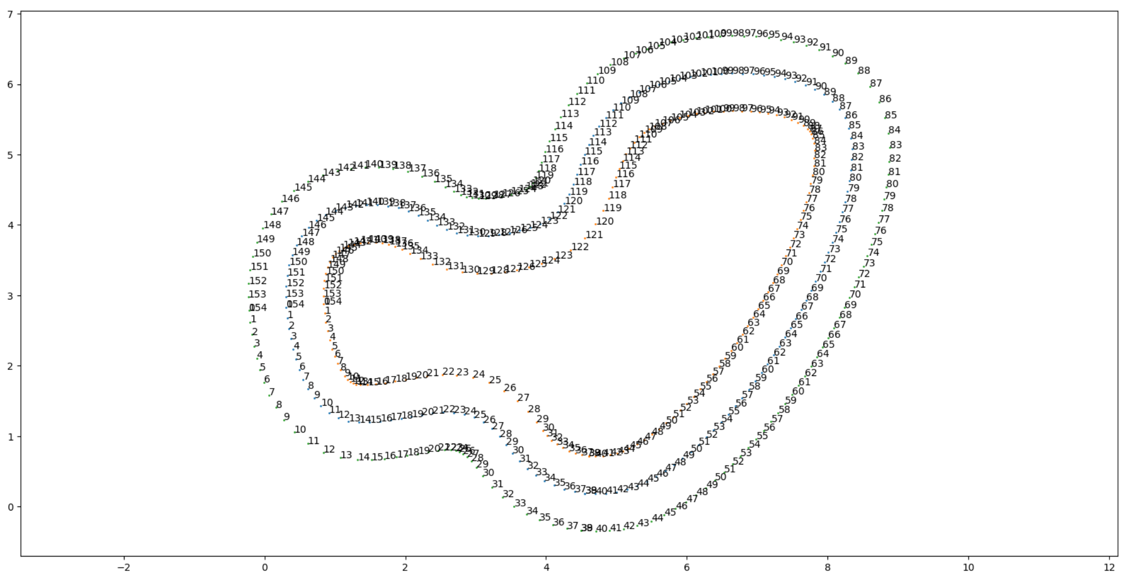
Now let's load the track:

[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.

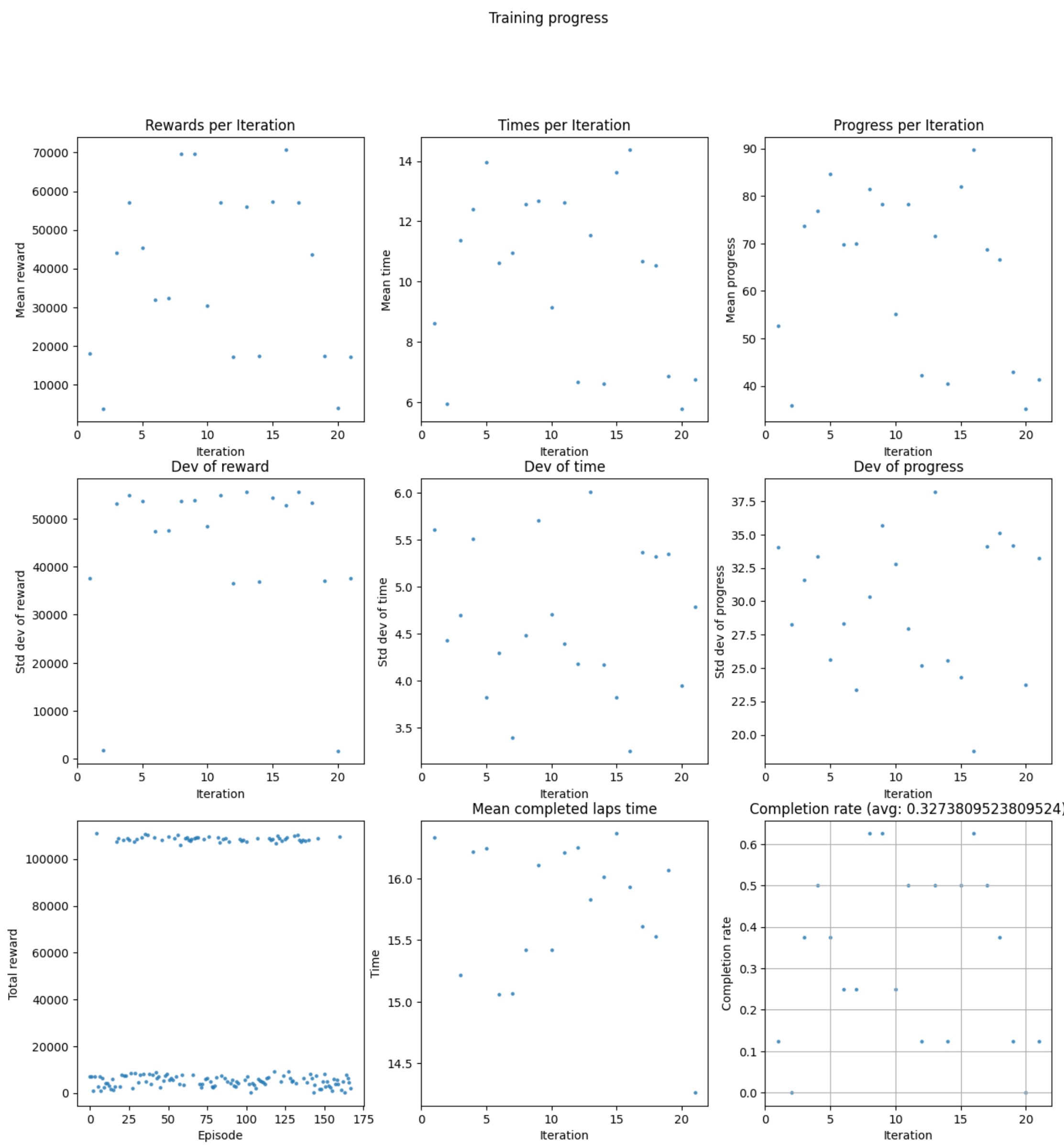
Mean completed lap times per iteration Once we have a model that completes laps reasonably often, we might want to know how fast the car gets around the track. This graph will show you that. I use it quite often when looking for a model to shave a couple more miliseconds. That said it has to go in pair 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]:
```python
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 =  167
Number of iterations =  21
```



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

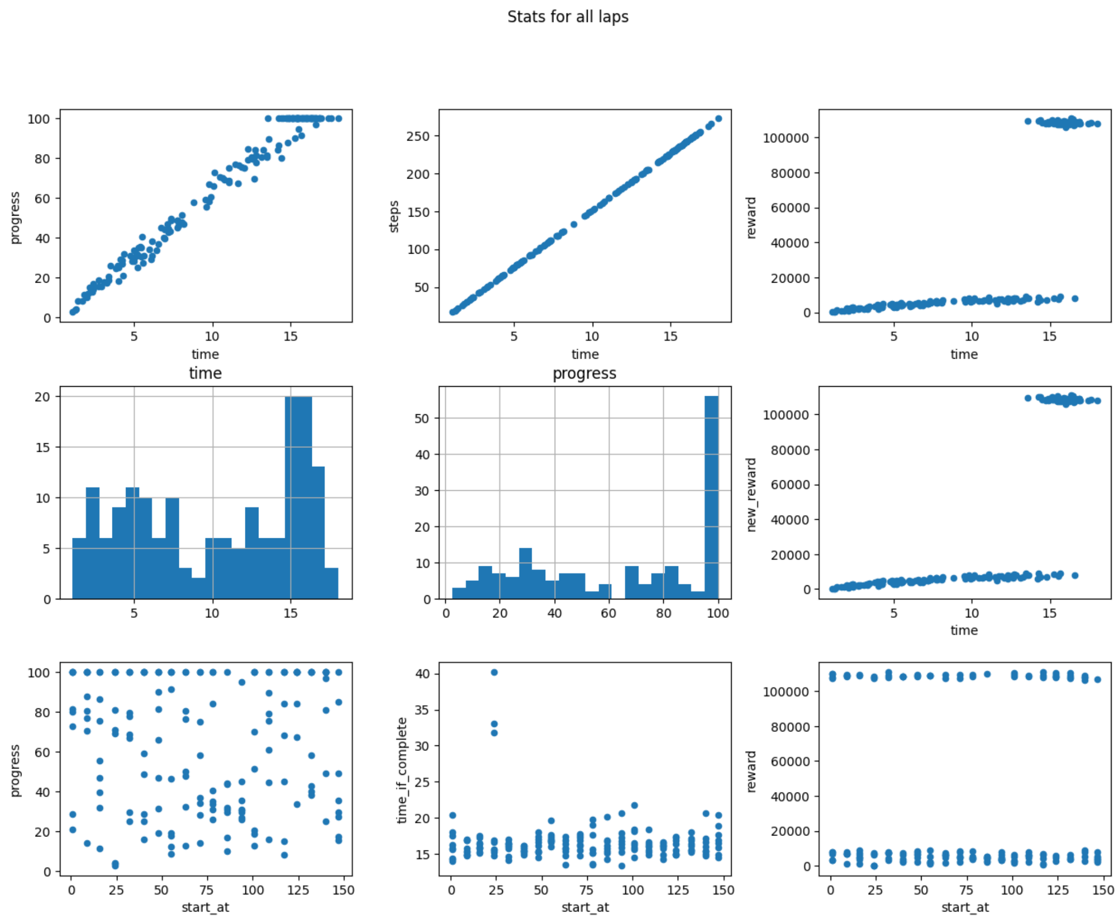
2.6.2 Stats for all laps

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

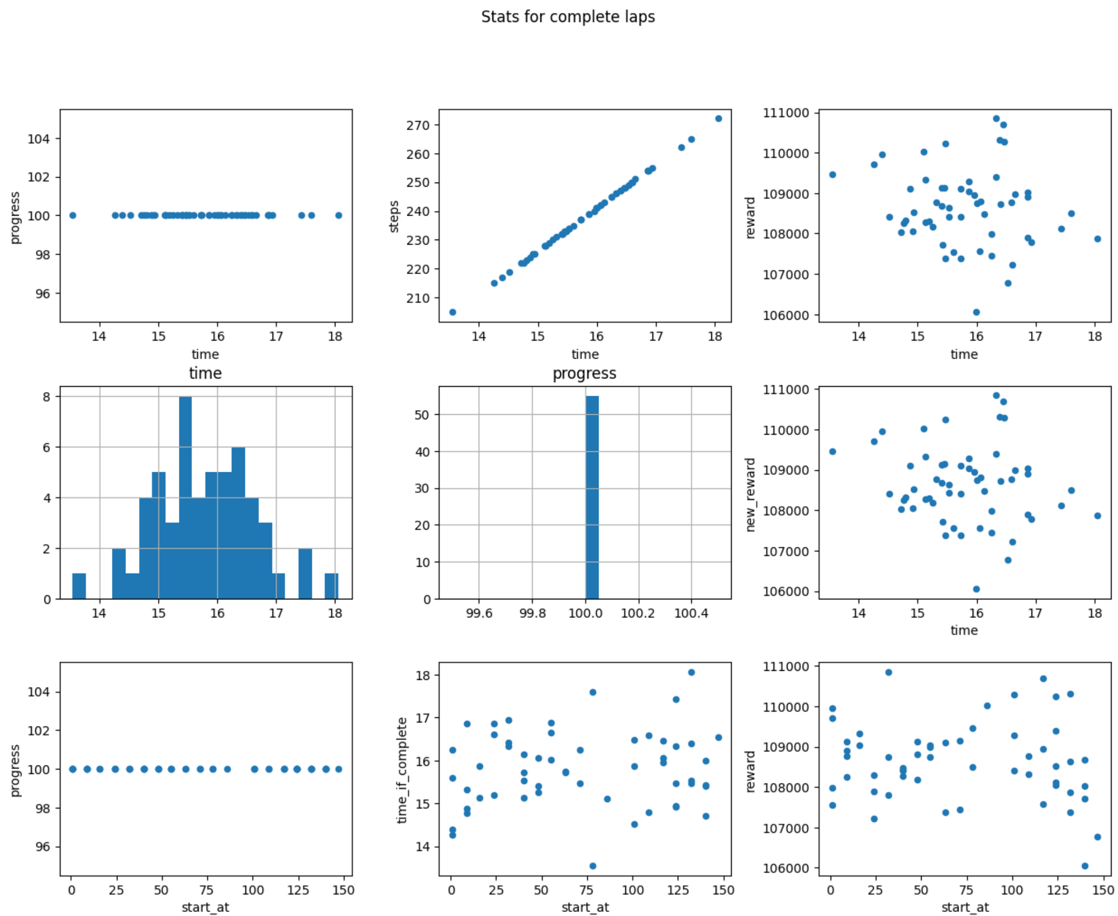



```
<Figure size 640x480 with 0 Axes>
```

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)`

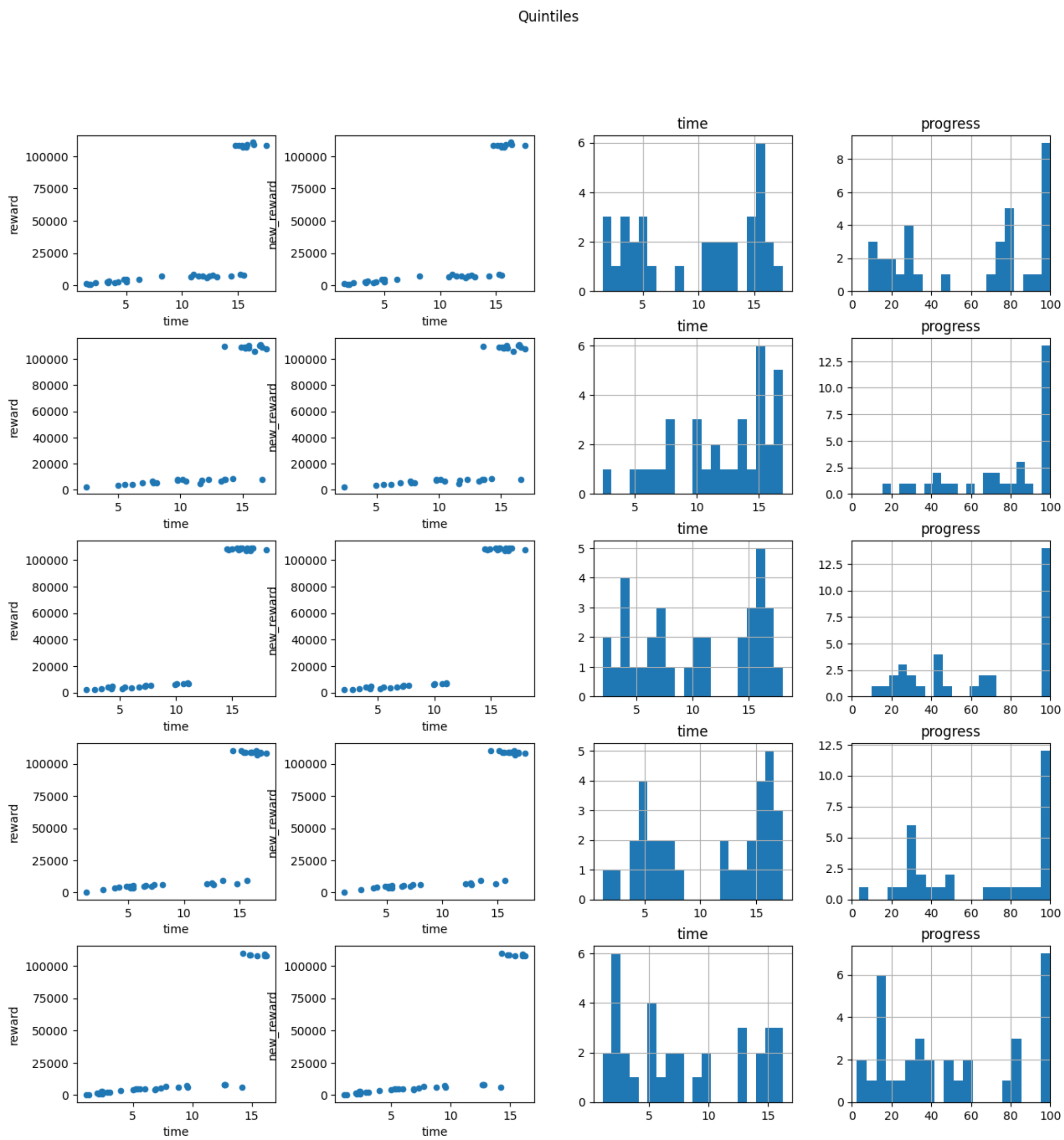
The idea is pretty simple - determine a way to cluster the data and provide that as the `category` parameter (alongside the count of groups available). In the default case we take advantage of the aggregated information to which quintile an episode belongs and thus build buckets each containing 20% of 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]:
```python
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
4	1	4	246	32	100.0	16.331	22.209090
35	5	35	248	117	100.0	16.458	21.425372
37	5	37	247	132	100.0	16.397	21.570888
133	17	133	248	101	100.0	16.472	22.005421
56	8	56	233	124	100.0	15.477	21.995032
131	17	131	228	86	100.0	15.108	21.654693
120	16	120	217	1	100.0	14.395	21.919209
160	21	160	215	1	100.0	14.262	22.033708
50	7	50	205	78	100.0	13.551	21.722861
76	10	76	246	124	100.0	16.327	22.244200

	new_reward	speed	reward	time_if_complete	reward_if_complete
4	110845.6816	1.857846	110845.6816	16.331	110845.6816
35	110695.8140	1.674435	110695.8140	16.458	110695.8140
37	110306.7725	1.731741	110306.7725	16.397	110306.7725
133	110280.9260	1.803185	110280.9260	16.472	110280.9260
56	110236.9012	1.891245	110236.9012	15.477	110236.9012
131	110028.0753	1.833816	110028.0753	15.108	110028.0753
120	109963.3317	1.945484	109963.3317	14.395	109963.3317
160	109714.9378	1.930605	109714.9378	14.262	109714.9378
50	109458.6875	2.066976	109458.6875	13.551	109458.6875
76	109397.2808	1.825813	109397.2808	16.327	109397.2808

	quintile	complete

```
4        1st         1
35       2nd         1
37       2nd         1
133      4th         1
56       2nd         1
131      4th         1
120      4th         1
160      5th         1
50       2nd         1
76       3rd         1
```

[13]:
```
# View five fastest complete laps
complete_ones.nsmallest(5, 'time')
```

[13]:
```
     iteration  episode  steps  start_at  progress    time       dist  \
50           7       50    205        78     100.0  13.551  21.722861
160         21      160    215         1     100.0  14.262  22.033708
120         16      120    217         1     100.0  14.395  21.919209
73          10       73    219       101     100.0  14.521  22.026358
98          13       98    222       140     100.0  14.714  22.304676

      new_reward     speed       reward  time_if_complete  reward_if_complete  \
50   109458.6875  2.066976  109458.6875            13.551         109458.6875
160  109714.9378  1.930605  109714.9378            14.262         109714.9378
120  109963.3317  1.945484  109963.3317            14.395         109963.3317
73   108415.1795  1.824703  108415.1795            14.521         108415.1795
98   108037.2761  1.925631  108037.2761            14.714         108037.2761

    quintile  complete
50       2nd         1
160      5th         1
120      4th         1
73       3rd         1
98       3rd         1
```

[14]:
```
# View five best rewarded completed laps
complete_ones.nlargest(5, 'reward')
```

[14]:
```
     iteration  episode  steps  start_at  progress    time       dist  \
4            1        4    246        32     100.0  16.331  22.209090
35           5       35    248       117     100.0  16.458  21.425372
37           5       37    247       132     100.0  16.397  21.570888
133         17      133    248       101     100.0  16.472  22.005421
56           8       56    233       124     100.0  15.477  21.995032

      new_reward     speed       reward  time_if_complete  reward_if_complete  \
4    110845.6816  1.857846  110845.6816            16.331         110845.6816
```

```
35    110695.8140  1.674435  110695.8140            16.458           110695.8140
37    110306.7725  1.731741  110306.7725            16.397           110306.7725
133   110280.9260  1.803185  110280.9260            16.472           110280.9260
56    110236.9012  1.891245  110236.9012            15.477           110236.9012

    quintile  complete
4        1st         1
35       2nd         1
37       2nd         1
133      4th         1
56       2nd         1
```

[15]:

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

[15]:

```
     iteration  episode  steps  start_at  progress    time       dist  \
4            1        4    246        32     100.0  16.331  22.209090
35           5       35    248       117     100.0  16.458  21.425372
37           5       37    247       132     100.0  16.397  21.570888
133         17      133    248       101     100.0  16.472  22.005421
56           8       56    233       124     100.0  15.477  21.995032

      new_reward     speed       reward  time_if_complete  reward_if_complete  \
4    110845.6816  1.857846  110845.6816            16.331         110845.6816
35   110695.8140  1.674435  110695.8140            16.458         110695.8140
37   110306.7725  1.731741  110306.7725            16.397         110306.7725
133  110280.9260  1.803185  110280.9260            16.472         110280.9260
56   110236.9012  1.891245  110236.9012            15.477         110236.9012

    quintile  complete
4        1st         1
35       2nd         1
37       2nd         1
133      4th         1
56       2nd         1
```

[16]:

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

[16]:

```
    iteration  episode  steps  start_at  progress    time       dist  \
4           1        4    246        32     100.0  16.331  22.209090
17          3       17    233       132     100.0  15.474  23.010944
18          3       18    232       140     100.0  15.403  22.513986
21          3       21    222         9     100.0  14.764  21.954131
24          4       24    247        32     100.0  16.408  22.749064
```

```
      new_reward     speed       reward  time_if_complete  reward_if_complete  \
4   110845.6816  1.857846  110845.6816            16.331         110845.6816
17  107380.6544  1.874335  107380.6544            15.474         107380.6544
18  108683.9970  1.922414  108683.9970            15.403         108683.9970
21  108251.5259  1.884189  108251.5259            14.764         108251.5259
24  108733.8595  1.732348  108733.8595            16.408         108733.8595

   quintile  complete
4       1st         1
17      1st         1
18      1st         1
21      1st         1
24      1st         1
```

[17]:
```python
# View information for a couple first episodes
simulation_agg.head()
```

[17]:
```
   iteration  episode  steps  start_at  progress    time       dist  \
0          1        0    217         1   79.9695  14.388  18.576526
1          1        1    173         9   76.8821  11.462  17.970916
2          1        2     28        16   11.3243   1.798   2.661552
3          1        3    163        24   69.1653  10.793  15.479347
4          1        4    246        32  100.0000  16.331  22.209090

     new_reward     speed       reward  time_if_complete  reward_if_complete  \
0     6962.5092  1.726221    6962.5092         17.991859         8706.455836
1     6975.3650  1.967919    6975.3650         14.908542         9072.807585
2     1043.9116  2.361429    1043.9116         15.877361         9218.332259
3     6880.7634  1.936442    6880.7634         15.604646         9948.288231
4   110845.6816  1.857846  110845.6816         16.331000       110845.681600

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

[18]:
```python
# 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]
```

[18]:
```
      iteration  episode  steps       x       y      yaw  steering_angle  \
1273          2       10      1  8.2684  4.3956  75.0599           28.89
```

1274	2	10	2	8.2683	4.3955	75.0983	16.22
1275	2	10	3	8.2693	4.4037	75.3984	7.61
1276	2	10	4	8.2709	4.4247	76.2905	29.05
1277	2	10	5	8.2718	4.4618	78.3496	-13.85
1278	2	10	6	8.2679	4.5028	81.4904	11.44
1279	2	10	7	8.2689	4.5588	83.2576	-8.95
1280	2	10	8	8.2771	4.6163	83.0857	28.50
1281	2	10	9	8.2849	4.6711	82.9922	17.61
1282	2	10	10	8.2810	4.7451	85.9031	22.44
1283	2	10	11	8.2622	4.8258	91.3430	-9.25
1284	2	10	12	8.2388	4.9257	95.7764	-14.03
1285	2	10	13	8.2337	5.0361	94.5198	-15.00
1286	2	10	14	8.2401	5.1470	91.3831	25.03
1287	2	10	15	8.2505	5.2418	88.7431	23.21
1288	2	10	16	8.2467	5.3652	89.8936	30.00
1289	2	10	17	8.2217	5.4847	95.1952	30.00
1290	2	10	18	8.1780	5.6077	103.5233	14.96
1291	2	10	19	8.1306	5.7124	108.5666	30.00
1292	2	10	20	8.0687	5.8239	113.6686	25.87
1293	2	10	21	8.0079	5.9052	119.0786	30.00
1294	2	10	22	7.9570	5.9614	122.8978	30.00
1295	2	10	23	7.8994	6.0122	127.5983	30.00
1296	2	10	24	7.8505	6.0403	132.5873	30.00
1297	2	10	25	7.7979	6.0628	138.2082	25.87
1298	2	10	26	7.7442	6.0725	144.9067	30.00
1299	2	10	27	7.6982	6.0771	150.2619	30.00
1300	2	10	28	7.6487	6.0794	155.5489	19.71
1301	2	10	29	7.5921	6.0779	161.2624	-2.23
1302	2	10	30	7.5326	6.0828	164.5936	-15.00
1303	2	10	31	7.4702	6.1015	164.4385	-11.74
1304	2	10	32	7.4021	6.1228	163.9757	-6.99
1305	2	10	33	7.3223	6.1510	162.8585	30.00
1306	2	10	34	7.2360	6.1715	164.0512	30.00
1307	2	10	35	7.1443	6.1762	168.8409	-3.90
1308	2	10	36	7.0743	6.1777	171.5315	19.94
1309	2	10	37	7.0039	6.1784	173.8940	-15.00
1310	2	10	38	6.9181	6.1788	175.7465	3.19
1311	2	10	39	6.8117	6.1805	177.1126	30.00
1312	2	10	40	6.7181	6.1667	-178.6547	21.23
1313	2	10	41	6.6100	6.1304	-170.7797	-14.11
1314	2	10	42	6.5220	6.0990	-166.9961	9.30
1315	2	10	43	6.4530	6.0752	-165.1466	4.20
1316	2	10	44	6.3830	6.0450	-162.5919	-9.31
1317	2	10	45	6.3236	6.0235	-161.8753	11.31
1318	2	10	46	6.2706	6.0058	-161.8198	14.14
1319	2	10	47	6.2122	5.9727	-158.7665	-8.04
1320	2	10	48	6.1522	5.9356	-155.6962	-15.00

1321	2	10	49	6.0867	5.9040	-155.2645	-15.00
1322	2	10	50	6.0098	5.8773	-156.9227	-13.32
1323	2	10	51	5.9395	5.8554	-158.5901	-15.00
1324	2	10	52	5.8778	5.8392	-160.2740	-11.98
1325	2	10	53	5.8041	5.8248	-162.7964	-15.00
1326	2	10	54	5.7326	5.8150	-165.4243	-15.00
1327	2	10	55	5.6553	5.8107	-168.8473	26.29
1328	2	10	56	5.5713	5.8008	-170.1764	7.42
1329	2	10	57	5.4687	5.7691	-167.2995	-15.00
1330	2	10	58	5.3791	5.7459	-164.9682	-11.54
1331	2	10	59	5.2795	5.7297	-167.0154	30.00
1332	2	10	60	5.1921	5.7149	-168.8581	-14.82
1333	2	10	61	5.1010	5.6928	-168.0232	30.00
1334	2	10	62	5.0022	5.6661	-166.5119	-2.03
1335	2	10	63	4.9192	5.6385	-164.9117	30.00
1336	2	10	64	4.8344	5.6052	-162.7902	30.00
1337	2	10	65	4.7677	5.5726	-160.1815	17.53
1338	2	10	66	4.7392	5.5507	-157.4547	19.10
1339	2	10	67	4.7012	5.5105	-152.0374	20.37
1340	2	10	68	4.6611	5.4506	-144.2876	30.00
1341	2	10	69	4.6365	5.3951	-136.8427	24.19
1342	2	10	70	4.6181	5.3352	-129.3892	11.22
1343	2	10	71	4.6071	5.2771	-122.8544	19.95
1344	2	10	72	4.6013	5.2157	-116.3116	13.98
1345	2	10	73	4.6066	5.1305	-106.7518	5.73
1346	2	10	74	4.6208	5.0404	-97.7415	-15.00
1347	2	10	75	4.6314	4.9434	-91.0744	-15.00
1348	2	10	76	4.6355	4.8306	-89.0124	-15.00
1349	2	10	77	4.6263	4.7020	-91.4249	-15.00
1350	2	10	78	4.6061	4.5791	-94.9992	-8.22
1351	2	10	79	4.5751	4.4528	-99.3180	-15.00
1352	2	10	80	4.5380	4.3332	-102.8732	-15.00
1353	2	10	81	4.4887	4.2131	-107.2563	-15.00
1354	2	10	82	4.4384	4.1187	-111.6029	-15.00
1355	2	10	83	4.3695	4.0141	-117.0687	-14.80
1356	2	10	84	4.2833	3.9090	-123.1909	16.98
1357	2	10	85	4.1959	3.8042	-126.5333	-2.53
1358	2	10	86	4.1146	3.6917	-126.6355	30.00
1359	2	10	87	4.0462	3.5918	-125.6743	30.00
1360	2	10	88	3.9924	3.5020	-123.7701	21.68
1361	2	10	89	3.9570	3.4231	-120.5697	-11.08
1362	2	10	90	3.9295	3.3532	-117.8352	-5.62
1363	2	10	91	3.8966	3.2939	-118.1599	15.15
1364	2	10	92	3.8706	3.2432	-117.9205	-15.00
1365	2	10	93	3.8451	3.1945	-118.1456	14.47

speed action reward done on_track progress closest_waypoint \

1273	0.60	−1	0.0000	0	True	0.6058	78
1274	0.60	−1	0.0010	0	True	0.6054	78
1275	1.45	−1	49.8005	0	True	0.6407	78
1276	3.49	−1	139.2706	0	True	0.7308	79
1277	0.60	−1	40.0010	0	True	0.8884	79
1278	1.97	−1	105.3834	0	True	1.0613	79
1279	0.60	−1	40.0010	0	True	1.3003	80
1280	2.56	−1	120.9906	0	True	1.5513	80
1281	3.57	−1	139.5816	0	True	1.7968	80
1282	2.56	−1	120.6466	0	True	2.1116	81
1283	2.94	−1	129.2773	0	True	2.4648	81
1284	1.94	−1	103.8254	0	True	2.8876	82
1285	0.71	−1	40.0010	0	True	3.3994	83
1286	4.00	−1	143.1333	0	True	3.9195	84
1287	3.41	−1	136.8067	0	True	4.3654	84
1288	2.69	−1	123.0505	0	True	4.9311	85
1289	0.60	−1	40.0010	0	True	5.4533	86
1290	2.38	−1	114.8061	0	True	6.0389	87
1291	0.60	−1	0.0010	0	True	6.5280	88
1292	0.60	−1	0.0010	0	True	7.0302	88
1293	1.02	−1	0.0010	0	True	7.4536	89
1294	0.60	−1	0.0010	0	True	7.7245	89
1295	0.60	−1	0.0010	0	True	8.0508	90
1296	0.60	−1	0.0010	0	True	8.2391	90
1297	0.60	−1	0.0010	0	True	8.4867	91
1298	0.60	−1	0.0010	0	True	8.7159	91
1299	0.60	−1	0.0010	0	True	8.8734	91
1300	1.56	−1	53.2431	0	True	9.0791	91
1301	2.28	−1	70.7591	0	True	9.3082	92
1302	0.60	−1	0.0010	0	True	9.5461	92
1303	2.48	−1	75.9227	0	True	9.8277	93
1304	3.87	−1	99.6330	0	True	10.1235	93
1305	2.94	−1	86.3731	0	True	10.4860	94
1306	0.60	−1	0.0010	0	True	10.8544	94
1307	0.60	−1	0.0010	0	True	11.2500	95
1308	2.56	−1	77.1521	0	True	11.5415	95
1309	2.96	−1	86.2017	0	True	11.8450	96
1310	2.54	−1	76.4632	0	True	12.2090	96
1311	2.86	−1	83.9471	0	True	12.6690	97
1312	0.60	−1	0.0010	0	True	13.0676	98
1313	0.60	−1	40.0010	0	True	13.5369	98
1314	1.15	−1	40.0010	0	True	13.9206	99
1315	0.72	−1	40.0010	0	True	14.2326	99
1316	0.67	−1	40.0010	0	True	14.5424	100
1317	0.60	−1	40.0010	0	True	14.8225	100
1318	2.20	−1	106.1794	0	True	15.0578	101
1319	0.60	−1	40.0010	0	True	15.3227	101

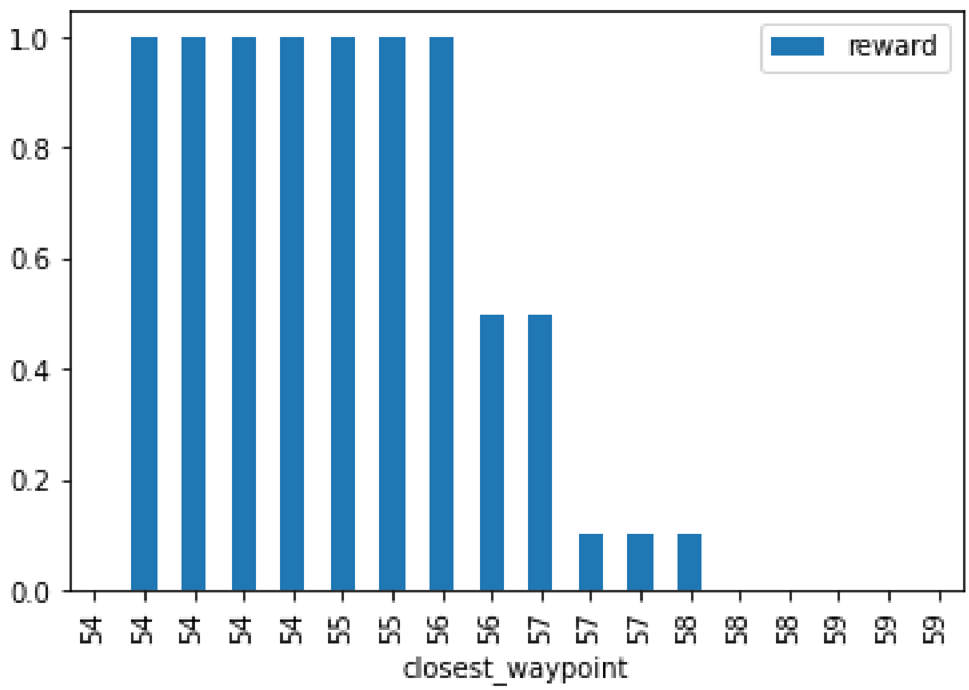
1320	3.71	−1	135.9603	0	True	15.6309	101
1321	1.04	−1	40.0010	0	True	15.9300	102
1322	0.60	−1	40.0010	0	True	16.3370	103
1323	0.60	−1	40.0010	0	True	16.7021	103
1324	4.00	−1	138.0667	0	True	16.9781	103
1325	1.08	−1	40.0010	0	True	17.3436	104
1326	1.40	−1	40.0010	0	True	17.6517	105
1327	4.00	−1	137.6667	0	True	18.0147	105
1328	3.49	−1	132.3798	0	True	18.3715	106
1329	0.61	−1	40.0010	0	True	18.8750	106
1330	3.49	−1	132.0904	0	True	19.3234	107
1331	0.60	−1	40.0010	0	True	19.7386	108
1332	3.14	−1	126.5140	0	True	20.1559	108
1333	0.60	−1	40.0010	0	True	20.5339	109
1334	1.72	−1	91.9474	0	True	20.9568	110
1335	0.60	−1	0.0010	0	True	21.2936	110
1336	0.60	−1	0.0010	0	True	21.6497	111
1337	0.60	−1	0.0010	0	True	21.9111	111
1338	0.60	−1	0.0010	0	True	22.0615	111
1339	2.09	−1	60.4663	0	True	22.2996	112
1340	0.60	−1	0.0010	0	True	22.5667	112
1341	0.60	−1	0.0010	0	True	22.8265	112
1342	0.90	−1	0.0010	0	True	23.0883	113
1343	1.77	−1	51.8502	0	True	23.3040	113
1344	1.94	−1	55.9028	0	True	23.5515	114
1345	4.00	−1	135.2667	0	True	23.8700	114
1346	2.11	−1	99.9904	0	True	24.2036	115
1347	4.00	−1	135.0000	0	True	24.5921	115
1348	4.00	−1	134.8667	0	True	25.0429	116
1349	2.27	−1	104.1567	0	True	25.5948	117
1350	1.12	−1	-59.9990	0	True	26.1152	117
1351	1.84	−1	-7.4963	0	True	26.6244	118
1352	1.13	−1	-59.9990	0	True	27.1121	119
1353	3.07	−1	42.6091	0	True	27.6163	120
1354	4.00	−1	54.0667	0	True	27.9027	120
1355	2.02	−1	16.4137	0	True	28.4200	121
1356	3.39	−1	47.3527	0	True	28.7921	122
1357	1.54	−1	25.3502	0	True	29.0927	122
1358	0.82	−1	-59.9990	0	True	29.6337	123
1359	0.60	−1	-59.9990	0	True	29.8224	123
1360	0.60	−1	-59.9990	0	False	30.1742	124
1361	0.60	−1	-59.9990	0	False	30.3814	124
1362	0.60	−1	-19.9990	0	False	30.4481	124
1363	0.60	−1	-59.9990	0	False	30.6545	124
1364	2.83	−1	36.0982	0	False	30.8223	125
1365	0.60	−1	-59.9990	1	False	30.9859	125

	track_len	tstamp	episode_status	pause_duration	new_reward
1273	23.12	168.07	prepare	0.0	0.0000
1274	23.12	168.136	in_progress	0.0	0.0010
1275	23.12	168.172	in_progress	0.0	49.8005
1276	23.12	168.262	in_progress	0.0	139.2706
1277	23.12	168.337	in_progress	0.0	40.0010
1278	23.12	168.393	in_progress	0.0	105.3834
1279	23.12	168.464	in_progress	0.0	40.0010
1280	23.12	168.53	in_progress	0.0	120.9906
1281	23.12	168.608	in_progress	0.0	139.5816
1282	23.12	168.666	in_progress	0.0	120.6466
1283	23.12	168.734	in_progress	0.0	129.2773
1284	23.12	168.781	in_progress	0.0	103.8254
1285	23.12	168.867	in_progress	0.0	40.0010
1286	23.12	168.927	in_progress	0.0	143.1333
1287	23.12	169.006	in_progress	0.0	136.8067
1288	23.12	169.067	in_progress	0.0	123.0505
1289	23.12	169.121	in_progress	0.0	40.0010
1290	23.12	169.202	in_progress	0.0	114.8061
1291	23.12	169.263	in_progress	0.0	0.0010
1292	23.12	169.333	in_progress	0.0	0.0010
1293	23.12	169.4	in_progress	0.0	0.0010
1294	23.12	169.44	in_progress	0.0	0.0010
1295	23.12	169.53	in_progress	0.0	0.0010
1296	23.12	169.604	in_progress	0.0	0.0010
1297	23.12	169.668	in_progress	0.0	0.0010
1298	23.12	169.738	in_progress	0.0	0.0010
1299	23.12	169.801	in_progress	0.0	0.0010
1300	23.12	169.875	in_progress	0.0	53.2431
1301	23.12	169.932	in_progress	0.0	70.7591
1302	23.12	169.999	in_progress	0.0	0.0010
1303	23.12	170.057	in_progress	0.0	75.9227
1304	23.12	170.137	in_progress	0.0	99.6330
1305	23.12	170.189	in_progress	0.0	86.3731
1306	23.12	170.242	in_progress	0.0	0.0010
1307	23.12	170.321	in_progress	0.0	0.0010
1308	23.12	170.383	in_progress	0.0	77.1521
1309	23.12	170.473	in_progress	0.0	86.2017
1310	23.12	170.543	in_progress	0.0	76.4632
1311	23.12	170.601	in_progress	0.0	83.9471
1312	23.12	170.675	in_progress	0.0	0.0010
1313	23.12	170.741	in_progress	0.0	40.0010
1314	23.12	170.784	in_progress	0.0	40.0010
1315	23.12	170.87	in_progress	0.0	40.0010
1316	23.12	170.934	in_progress	0.0	40.0010
1317	23.12	171.002	in_progress	0.0	40.0010
1318	23.12	171.039	in_progress	0.0	106.1794

1319	23.12	171.139	in_progress	0.0	40.0010
1320	23.12	171.196	in_progress	0.0	135.9603
1321	23.12	171.266	in_progress	0.0	40.0010
1322	23.12	171.341	in_progress	0.0	40.0010
1323	23.12	171.394	in_progress	0.0	40.0010
1324	23.12	171.469	in_progress	0.0	138.0667
1325	23.12	171.526	in_progress	0.0	40.0010
1326	23.12	171.59	in_progress	0.0	40.0010
1327	23.12	171.672	in_progress	0.0	137.6667
1328	23.12	171.737	in_progress	0.0	132.3798
1329	23.12	171.807	in_progress	0.0	40.0010
1330	23.12	171.864	in_progress	0.0	132.0904
1331	23.12	171.929	in_progress	0.0	40.0010
1332	23.12	172.002	in_progress	0.0	126.5140
1333	23.12	172.068	in_progress	0.0	40.0010
1334	23.12	172.124	in_progress	0.0	91.9474
1335	23.12	172.191	in_progress	0.0	0.0010
1336	23.12	172.271	in_progress	0.0	0.0010
1337	23.12	172.327	in_progress	0.0	0.0010
1338	23.12	172.389	in_progress	0.0	0.0010
1339	23.12	172.47	in_progress	0.0	60.4663
1340	23.12	172.533	in_progress	0.0	0.0010
1341	23.12	172.604	in_progress	0.0	0.0010
1342	23.12	172.676	in_progress	0.0	0.0010
1343	23.12	172.736	in_progress	0.0	51.8502
1344	23.12	172.804	in_progress	0.0	55.9028
1345	23.12	172.85	in_progress	0.0	135.2667
1346	23.12	172.938	in_progress	0.0	99.9904
1347	23.12	172.996	in_progress	0.0	135.0000
1348	23.12	173.058	in_progress	0.0	134.8667
1349	23.12	173.135	in_progress	0.0	104.1567
1350	23.12	173.202	in_progress	0.0	-59.9990
1351	23.12	173.27	in_progress	0.0	-7.4963
1352	23.12	173.329	in_progress	0.0	-59.9990
1353	23.12	173.404	in_progress	0.0	42.6091
1354	23.12	173.463	in_progress	0.0	54.0667
1355	23.12	173.535	in_progress	0.0	16.4137
1356	23.12	173.598	in_progress	0.0	47.3527
1357	23.12	173.666	in_progress	0.0	25.3502
1358	23.12	173.723	in_progress	0.0	-59.9990
1359	23.12	173.802	in_progress	0.0	-59.9990
1360	23.12	173.86	in_progress	0.0	-59.9990
1361	23.12	173.934	in_progress	0.0	-59.9990
1362	23.12	173.998	in_progress	0.0	-19.9990
1363	23.12	174.062	in_progress	0.0	-59.9990
1364	23.12	174.127	in_progress	0.0	36.0982
1365	23.12	174.202	off_track	0.0	-59.9990

2.8 Analyze the reward distribution for your reward function

[46]:
```python
# 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

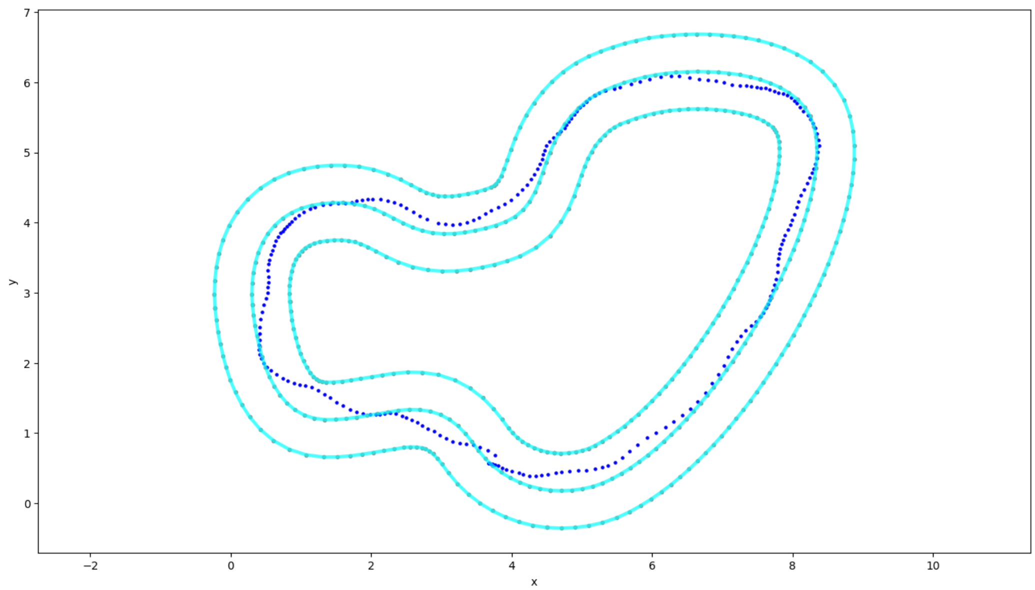
NOTE: at some point in the past in a single episode the car could go around multiple laps, the episode was terminated when car completed 1000 steps. Currently one episode has at most one lap. This explains why you can see multiple laps in an episode plotted below.

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.

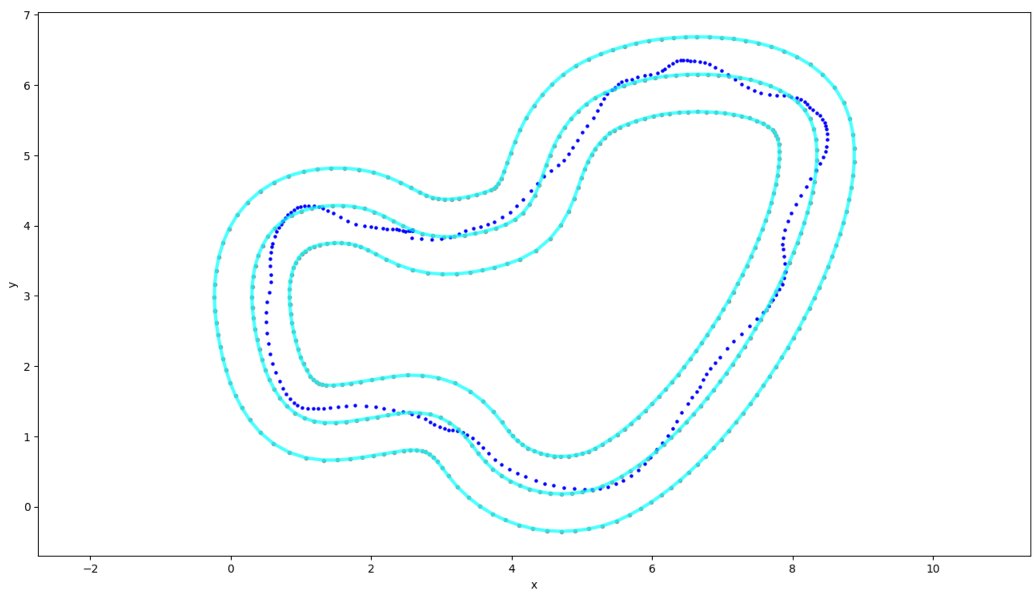
Graphs below give you a chance to look deeper into your car's behaviour on track.

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.

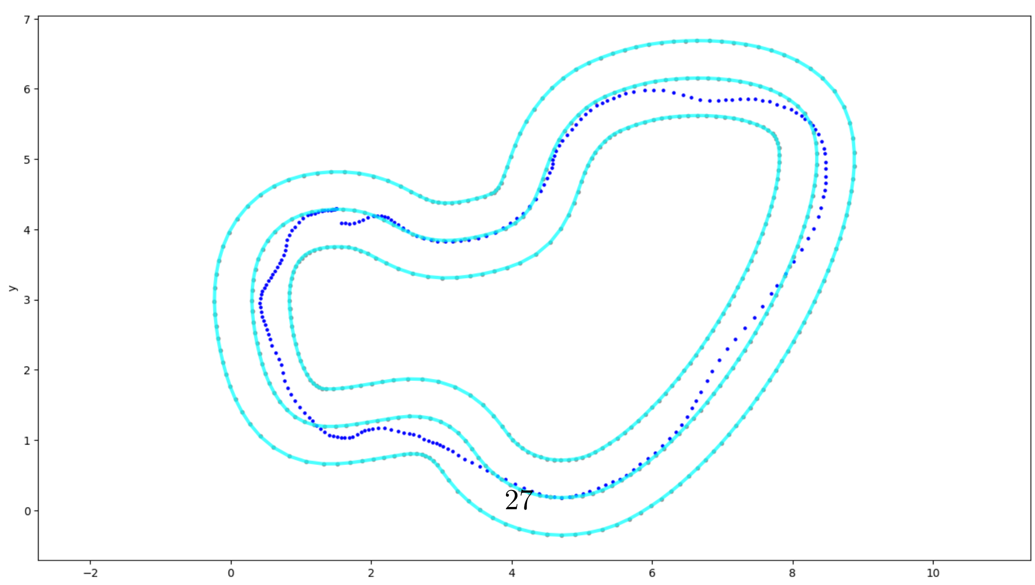
[19]:
```
# Some examples:
# highest reward for complete laps:
# episodes_to_plot = complete_ones.nlargest(3,'reward')

# 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

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

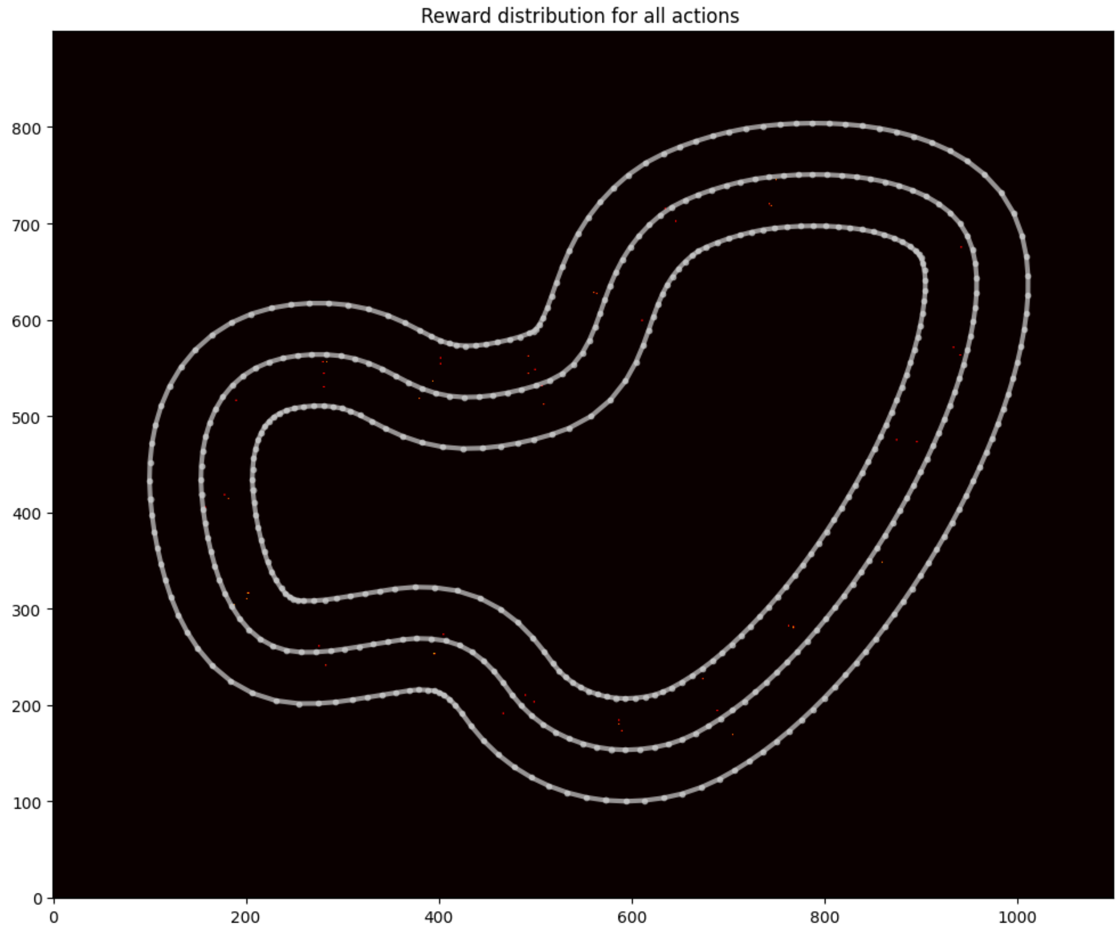
2.8.2 Plot a heatmap of rewards for current training.

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.

[20]:
```
#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)
```

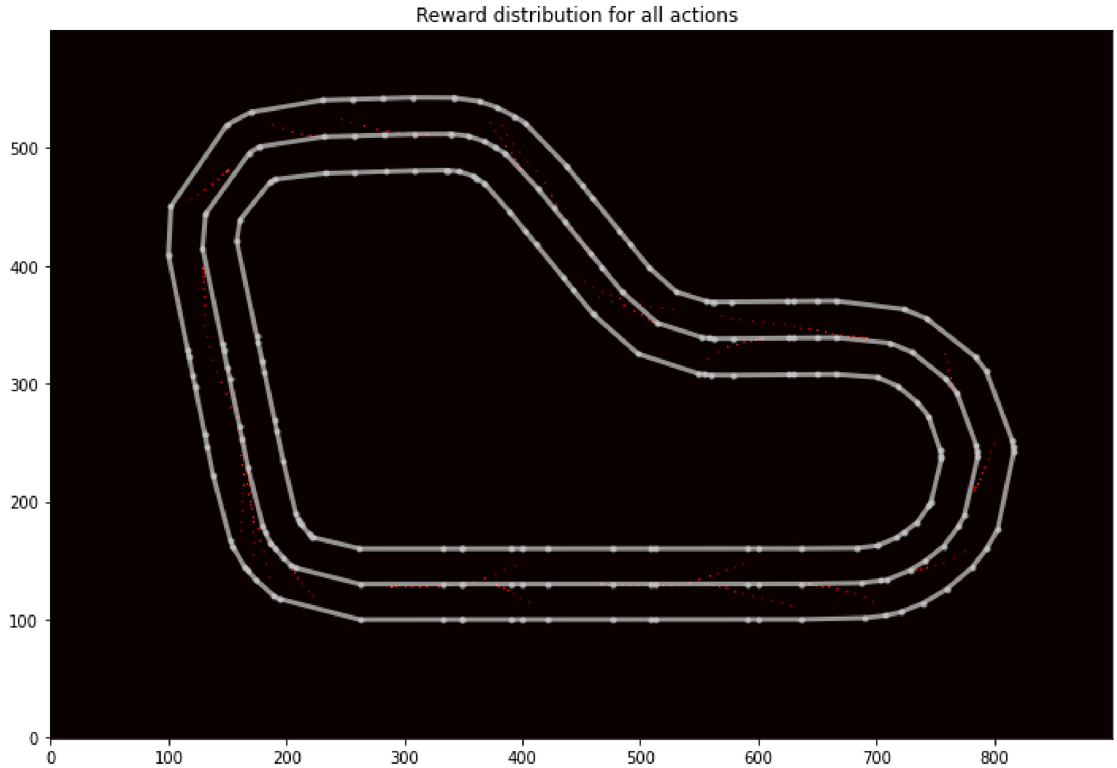


<Figure size 640x480 with 0 Axes>

2.8.3 Plot a particular iteration

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

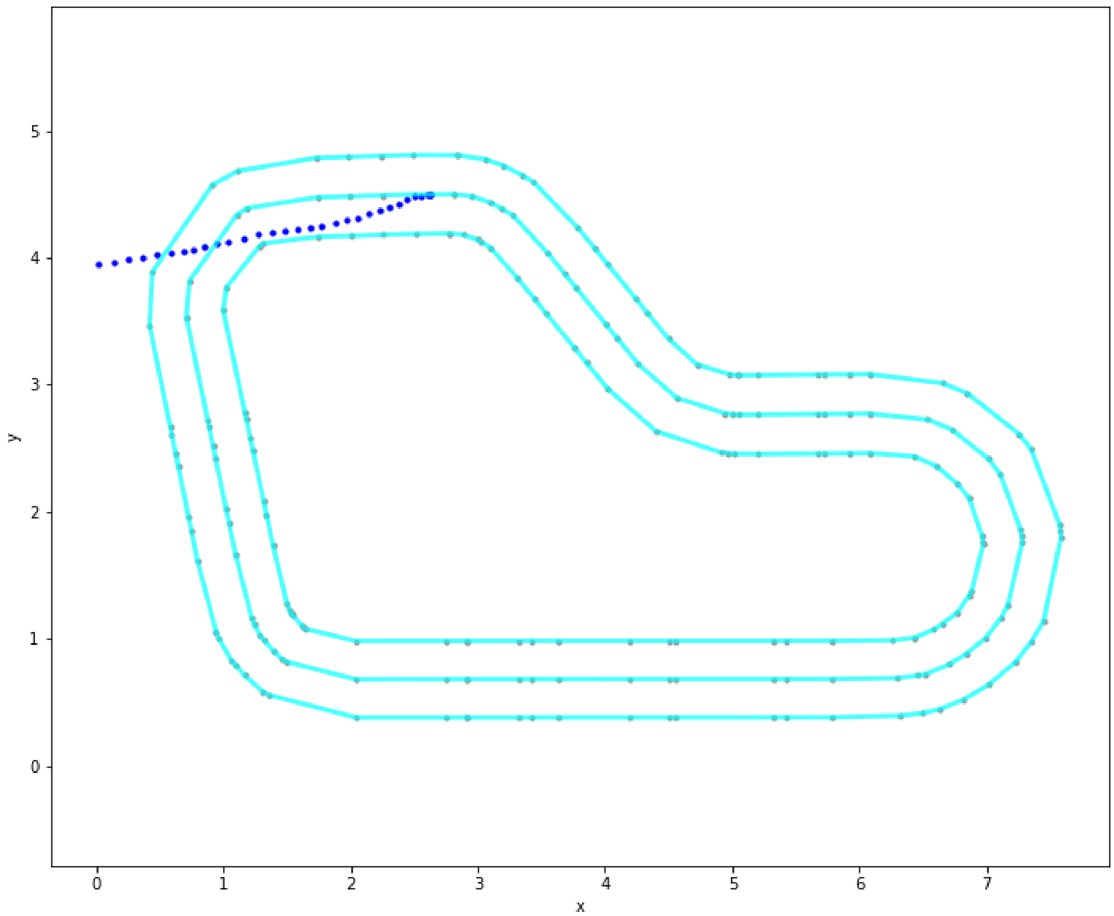
[49]:
```
#If you'd like some other colour criterion, you can add
#a value_field parameter and specify a different column
iteration_id = 3

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



<Figure size 432x288 with 0 Axes>

2.8.4 Path taken in a particular episode

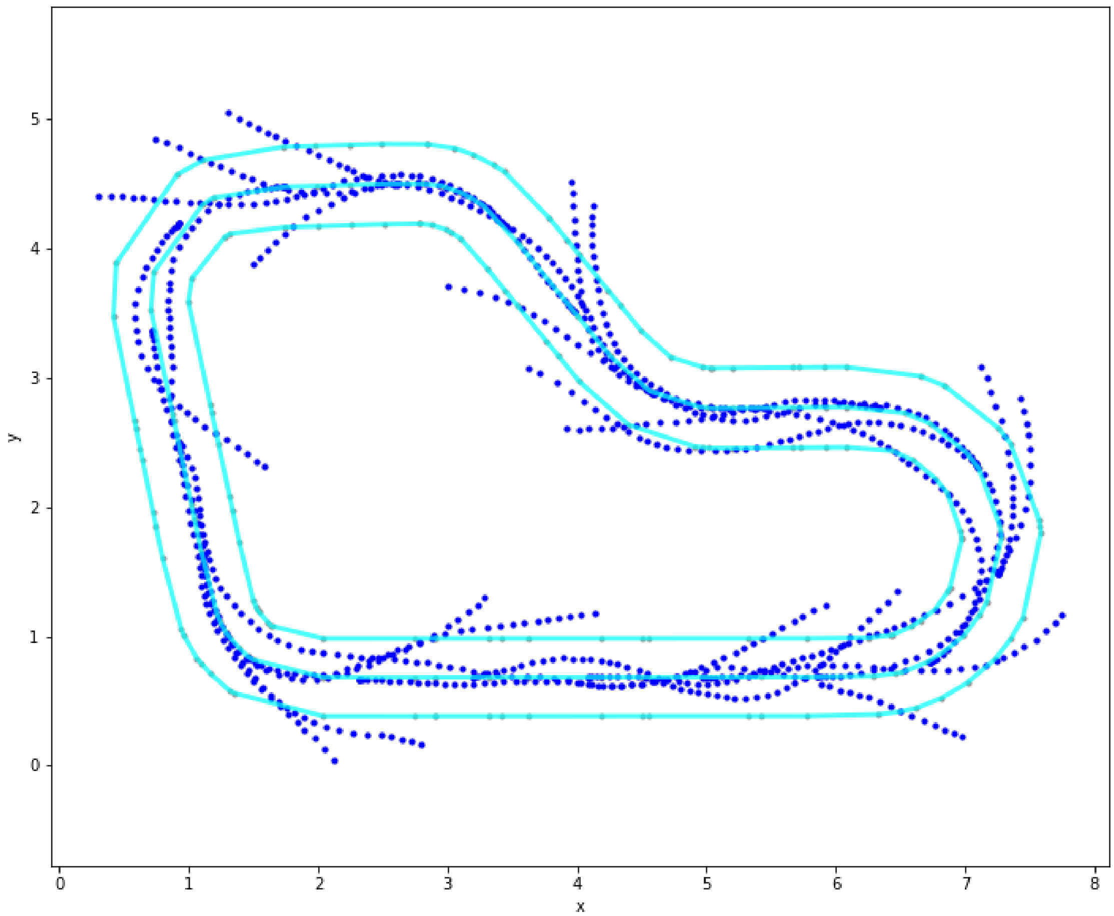
[50]:
```
episode_id = 12

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




```
<Figure size 432x288 with 0 Axes>
```

2.8.5 Path taken in a particular iteration

[51]:
```
iteration_id = 10

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



<Figure size 432x288 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:

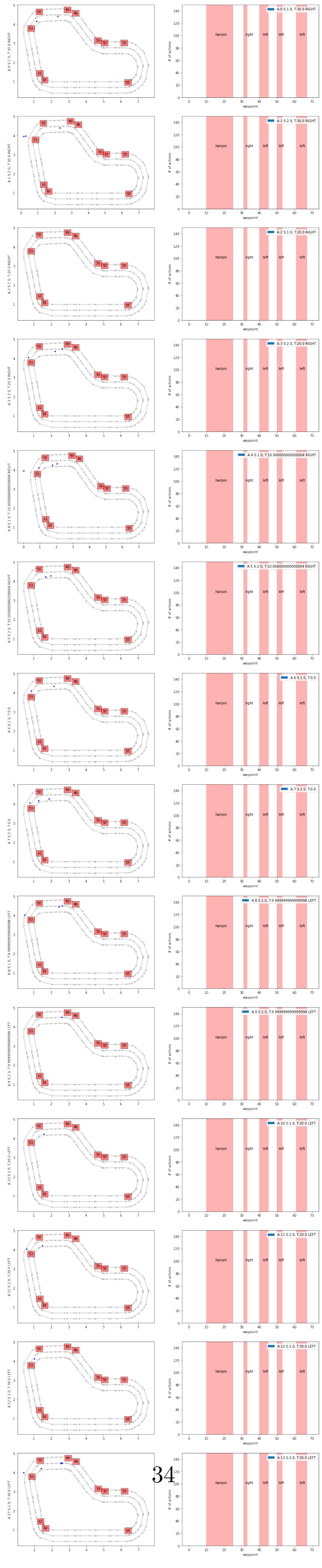
[52]:
```
track_breakdown.keys()
```

[52]:
```
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).

[53]:
```
abu.action_breakdown(df, track, track_breakdown=track_breakdown.
 ↪get('reinvent2018'), episode_ids=[12])
```


A:0 S:1.0, T:30.0 RIGHT
A:1 S:2.0, T:30.0 RIGHT
A:2 S:1.0, T:20.0 RIGHT
A:3 S:2.0, T:20.0 RIGHT
A:4 S:1.0, T:10.000000000000004 RIGHT
A:5 S:2.0, T:10.000000000000004 RIGHT
A:6 S:1.0, T:0.0
A:7 S:2.0, T:0.0
A:8 S:1.0, T:9.999999999999998 LEFT
A:9 S:2.0, T:9.999999999999998 LEFT
A:10 S:1.0, T:20.0 LEFT
A:11 S:2.0, T:20.0 LEFT
A:12 S:1.0, T:30.0 LEFT
A:13 S:2.0, T:30.0 LEFT
hairpin
right
left
left
left
# of actions
waypoint

```
&lt;Figure size 432x288 with 0 Axes&gt;
```

[ ]:
```
```

[ ]:
```
```

[ ]:
```
```