DeepRacer Car

University at Buffalo The State University of New York

Motivation

The AWS Deepracer is a complex environment for reinforcement learning. I can learn how the agent training with complex action space and reward function. It is also a good practice for learning autonomous vehicle. The most important is that the model we get from training can apply to the physical agent. This closely resembles a real-world use case.

For this project, we also need to process this system on local machine. This is a good way for us to get familiar with Docker, Cuda, Gazebo, etc, if we did not use these things before. We can find different repositories for the locally training. If you have time, try different repositories is also a chance to know how they organize the applications and what they modified for the Deepracer.

Agent: RC car with camera
Action: Go forward, turn left, turn right

Action Space

Maximum steering angle

30 degrees

Max values are between 1 and 30.

Steering angle granularity

3

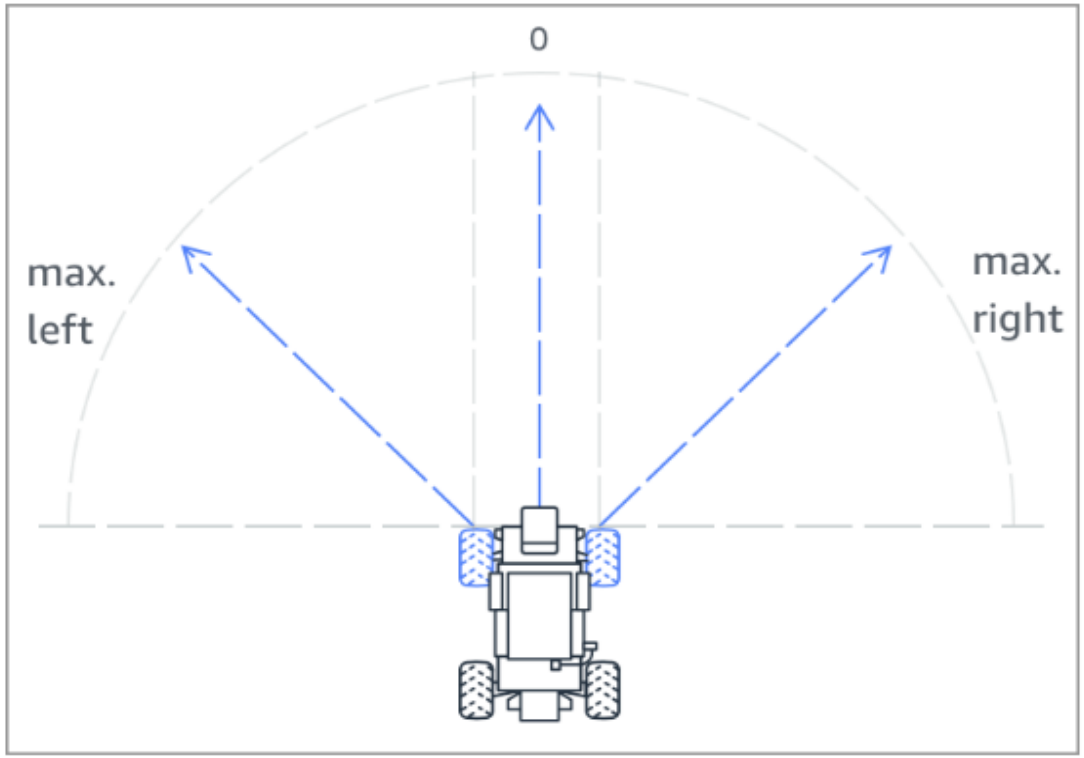
Maximum speed

1 m/s

Select values between 0.1 and 4.

Speed granularity

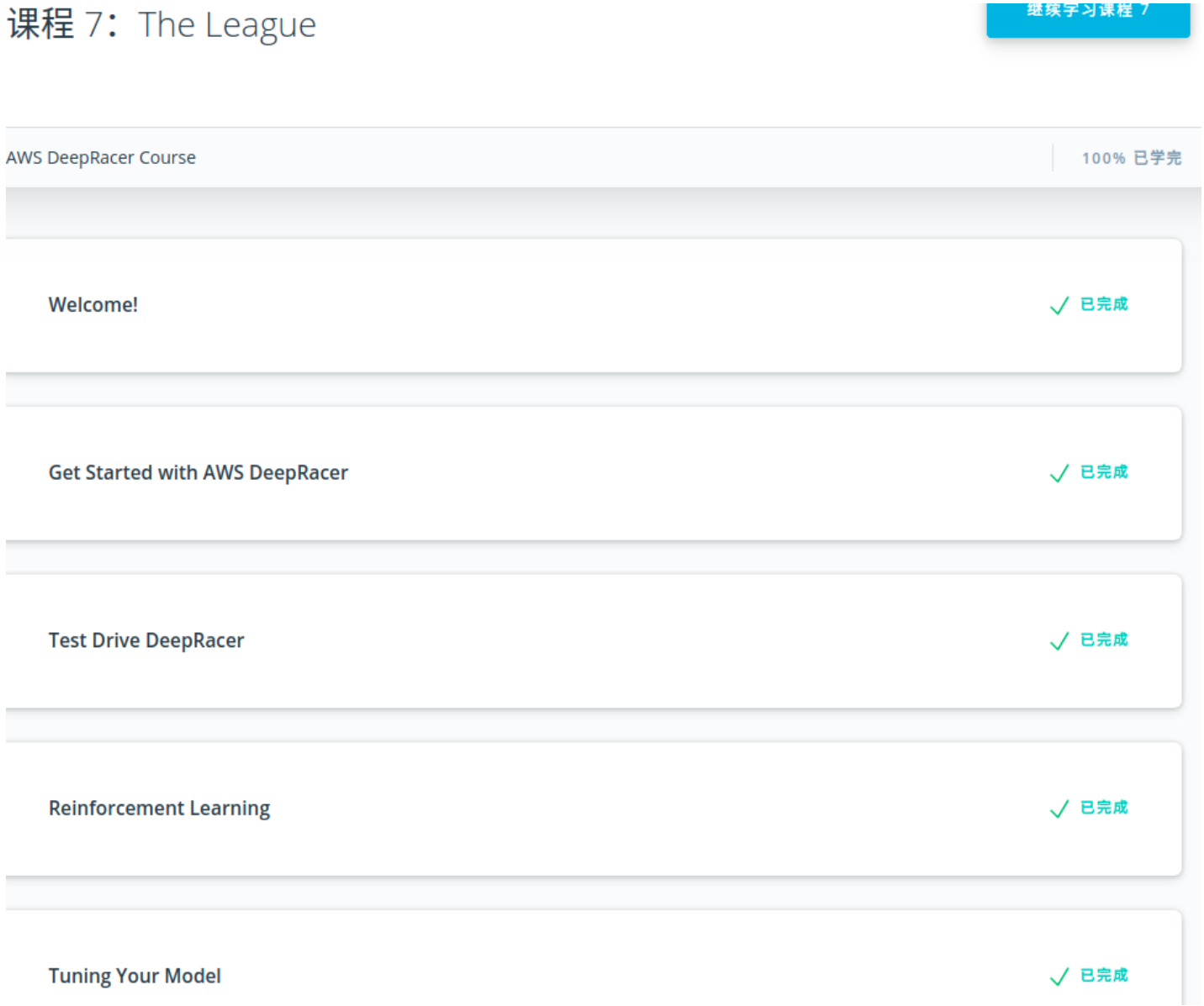
2



Action list

Action number	Steering angle	Speed
0	-30 degrees	0.5 m/s
1	-30 degrees	1 m/s
2	0 degrees	0.5 m/s
3	0 degrees	1 m/s
4	30 degrees	0.5 m/s
5	30 degrees	1 m/s

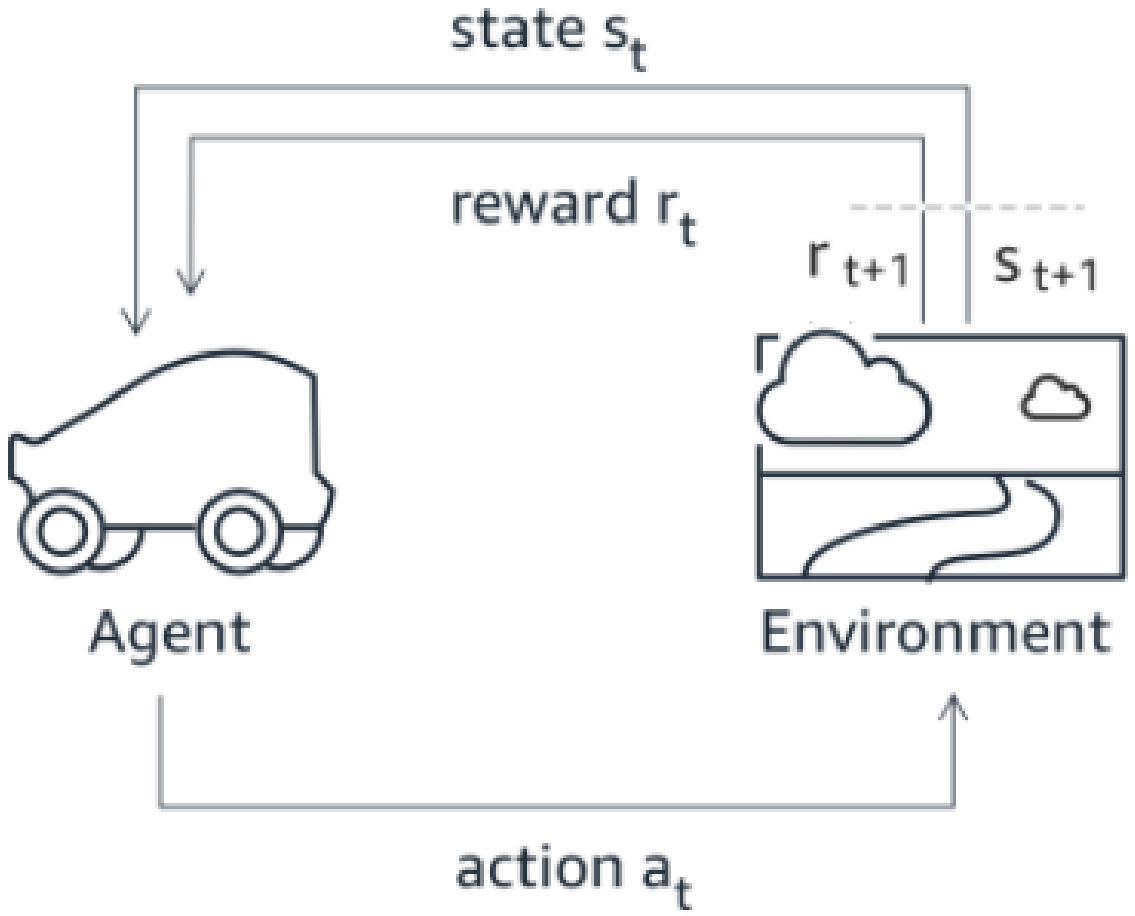
课程 7：The League



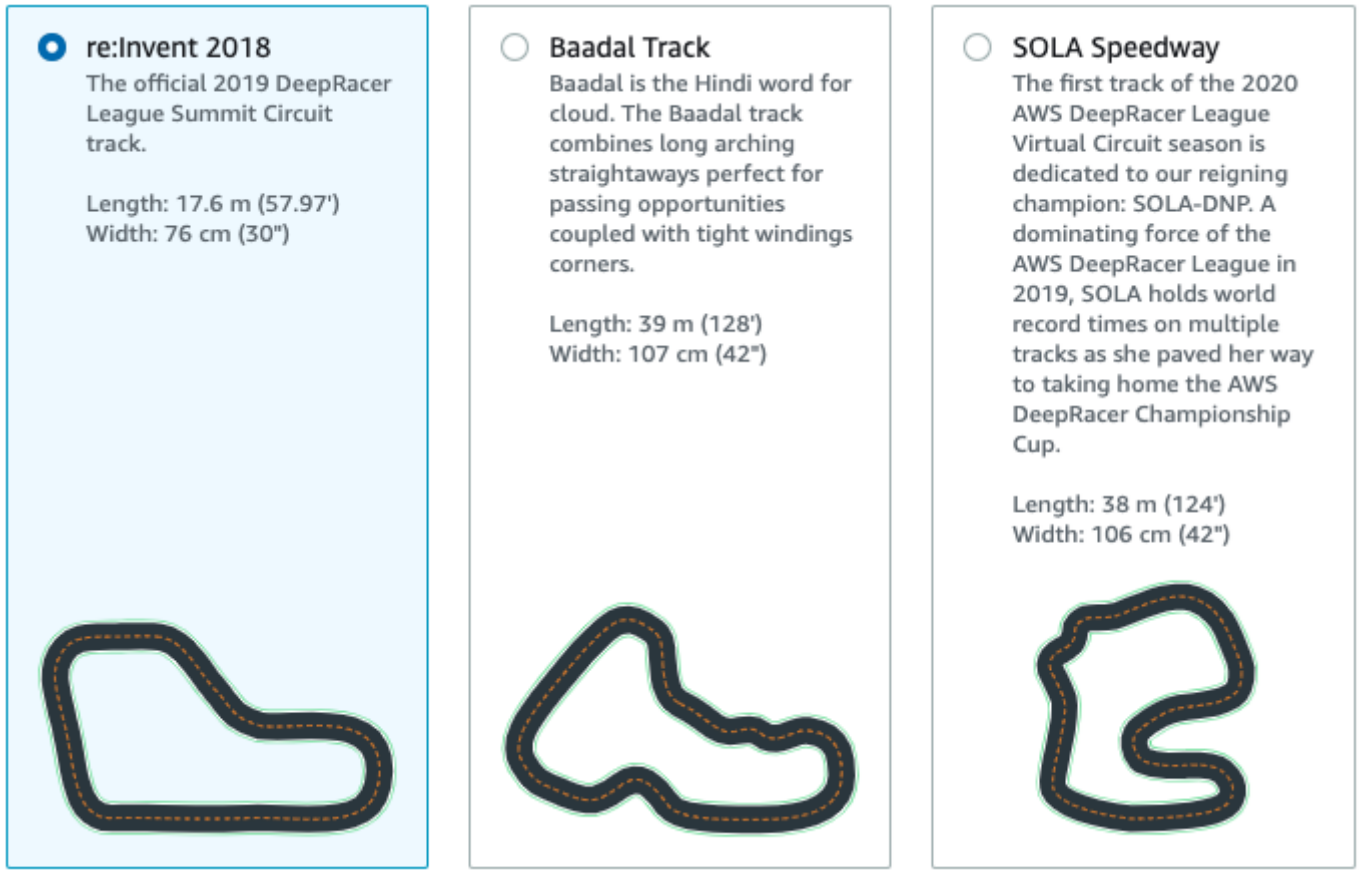
The course from Udacity provides good instructions for the deepracer car. It gives basic knowledges about RL.

It also described the main contents of the training steps and details about the hyperparameters and reward function.

Envrionment



In reinforcement learning for AWS DeepRacer, an **agent** (vehicle) learns from an **environment** (a track) by interacting with it and receiving rewards for performing specific actions.



We can choose different tracks as the environment.
The state is the position that the car on the track, and the image from There is a center lane on the track to help locate the car.
There are two conditions can be considered as finish. One is complete the track and the other one is drive out of the track.

University at Buffalo The State University of New York

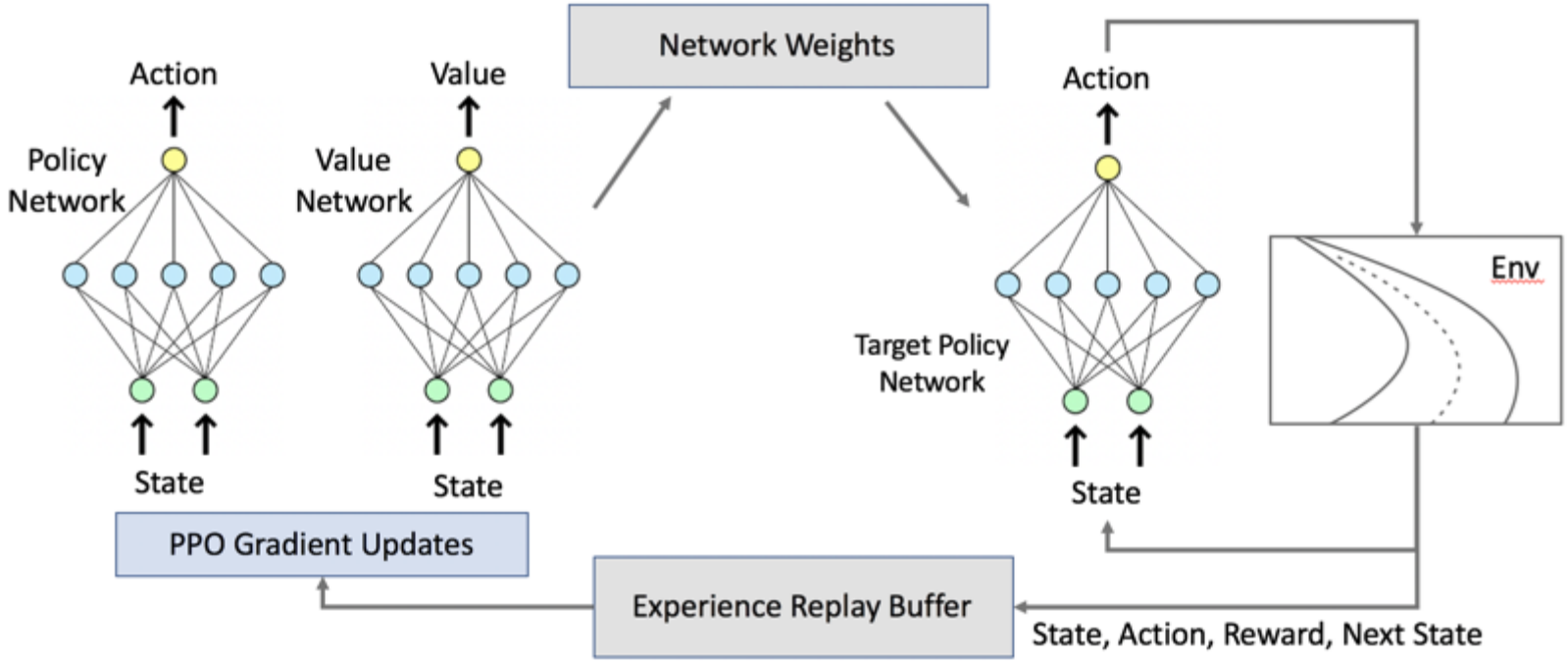
First Try before Checkpoint

Code editor | Reward function examples | Reset | Validate

```python
def reward_function(params):
    '''
    Example of rewarding the agent to follow center line
    '''

    # Read input parameters
    track_width = params['track_width']
    distance_from_center = params['distance_from_center']

    # Calculate 3 markers that are at varying distances away from the center line
    marker_1 = 0.1 * track_width
    marker_2 = 0.25 * track_width
    marker_3 = 0.5 * track_width

    # Give higher reward if the car is closer to center line and vice versa
    if distance_from_center <= marker_1:
        reward = 1.0
    elif distance_from_center <= marker_2:
        reward = 0.5
    elif distance_from_center <= marker_3:
        reward = 0.1
    else:
        reward = 1e-3  # likely crashed/ close to off track

    return float(reward)
```

This is the default reward function that helps agent to learn. It use the center line as the reference. The nearer to the center line the more rewards it will get.

We can modify it to gain better performance.

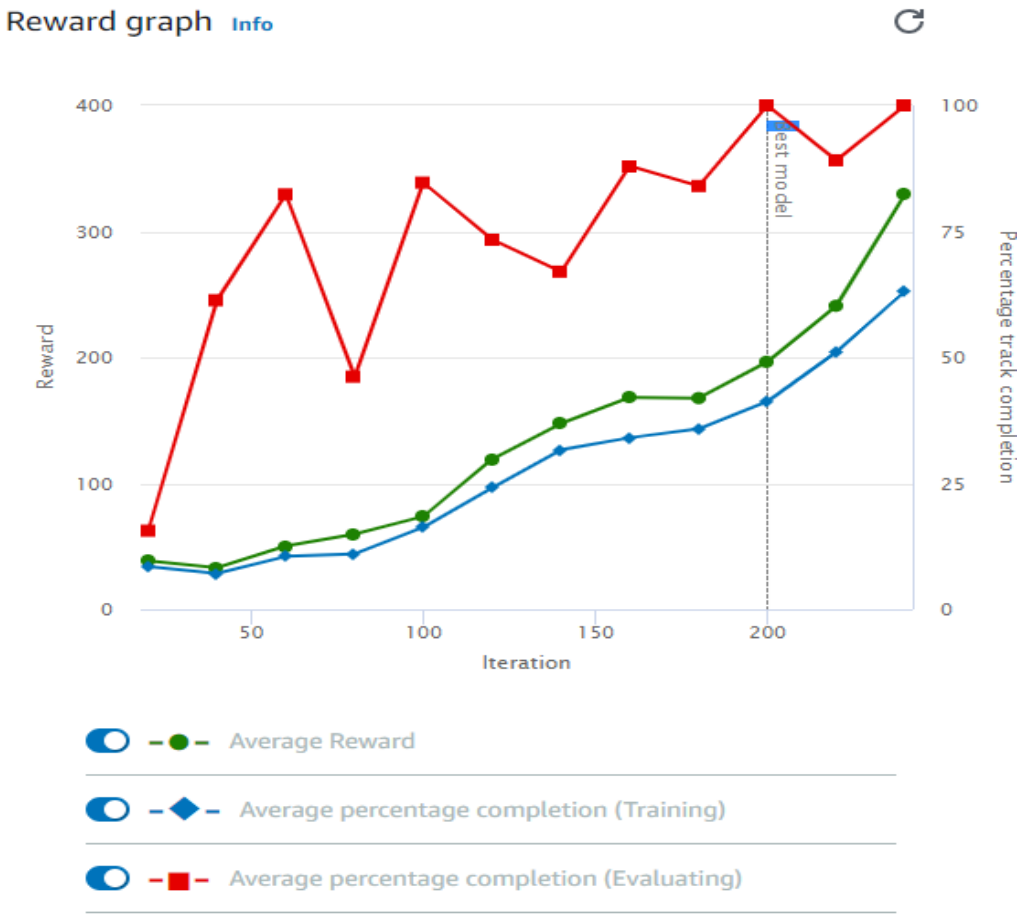


The default algorithm in DeepRacer is Proximal Policy Optimization algorithm.

PPO uses two neural networks during training: a policy network and a value network. The policy network (also called actor network) decides which action to take given an image as input. The value network (also called critic network) estimates the cumulative reward we are likely to get given the image as input. Only the policy network interacts with the simulator and gets deployed to the real agent, namely an AWS DeepRacer vehicle.

Hyperparameters of PPO

- **Gradient descent batch size:** The number recent vehicle experiences sampled at random from an experience buffer and used for updating the underlying deep-learning neural network weights.
- **Number of epochs:** The number of passes through the training data to update the neural network weights during gradient descent.
- **Learning rate:** During each update, a portion of the new weight can be from the gradient-descent (or ascent) contribution and the rest from the existing weight value.
- **Entropy:**A degree of uncertainty used to determine when to add randomness to the policy distribution.
- **Discount factor:** A factor specifies how much of the future rewards contribute to the expected reward.
- **Loss type:** Type of the objective function used to update the network weights.
- **Number of experience episodes between each policy-updating iteration:** The size of the experience buffer used to draw training data from for learning policy network weights.



Trial	Time	Trial results (% track completed)	Status
1	00:00:29.562	100%	Lap complete
2	00:00:15.609	50%	Off track
3	00:00:27.667	100%	Lap complete

Here is the result from the training using default rewards function and PPO algorithm. The result is pretty good and I even did not finish the training with some issues on the AWS. I setted training time to 1 hour and the training just ran for around half an hour, then it stopped and return error. The guide said I can still evaluate my result. So I got the above evaluation. Two run completed and one failed, not bad.

Install on Local

Repository From ARCC

- The recommended OS is Ubuntu 18.04, but I use 16.04 and it works
- Nvidia GPU needed
- CUDA and CUDNN
- Docker Docker compose and Nvidia-Docker
- AWS-cli(used for interaction with AWS)
- vncviewer(simulation visualization)

https://github.com/ARCC-RACE/deepracer-for-dummies.git
Git clone the repository from above link.
Then go into the folder and run the script init.sh for initialization.
Setup aws-cli(https://www.youtube.com/watch?v=FOK5BPy30HQ)

Edit the reward function in the deepracer-for-dummies/docker/volumes/minio/bucket/custom_files/reward.py file.

reward.py (~/deepracer-for-dummies/docker/volumes/minio/bucket/custom_files) - gedit

Open Save

```
def reward_function(params):
    '''
    Example of penalize steering, which helps mitigate zig-zag behaviors
    '''

    # Read input parameters
    distance_from_center = params['distance_from_center']
    track_width = params['track_width']
    steering = abs(params['steering_angle']) # Only need the absolute steering angle

    # Calculate 3 marks that are farther and father away from the center line
    marker_1 = 0.1 * track_width
    marker_2 = 0.25 * track_width
    marker_3 = 0.5 * track_width

    # Give higher reward if the car is closer to center line and vice versa
    if distance_from_center <= marker_1:
        reward = 1
    elif distance_from_center <= marker_2:
        reward = 0.5
    elif distance_from_center <= marker_3:
        reward = 0.1
    else:
        reward = 1e-3  # likely crashed/ close to off track

    # Steering penality threshold, change the number based on your action space setting
    ABS_STEERING_THRESHOLD = 15

    # Penalize reward if the car is steering too much
    if steering > ABS_STEERING_THRESHOLD:
        reward *= 0.8

    return float(reward)
```

Python Tab Width: 8 Ln 1, Col 1 INS

Change the track in `deepracer-for-dummies/docker/.env`.

.env (~/deepracer-for-dummies/docker) - gedit

Open Save

reward.py .env

```
WORLD_NAME=China_track
LOCAL_ENV_VAR_JSON_PATH=env_vars.json
MINIO_ACCESS_KEY=minio
MINIO_SECRET_KEY=miniokey
AWS_ACCESS_KEY_ID=minio
AWS_SECRET_ACCESS_KEY=miniokey
AWS_DEFAULT_REGION=us-east-1
S3_ENDPOINT_URL=http://minio:9000
ROS_AWS_REGION=us-east-1
AWS_REGION=us-east-1
MODEL_S3_PREFIX=rl-deepracer-sagemaker
MODEL_S3_BUCKET=bucket
LOCAL=True
MARKOV_PRESET_FILE=deepracer.py
XAUTHORITY=/root/.Xauthority
DISPLAY_N=:0
```

Plain Text Tab Width: 8 Ln 1, Col 1 INS

Adjust the hyperparameters list in the deepracer-for-dummies/docker/volumes/minio/bucket/custom_files/model_metadata.json file.

Make sure to update the action indices as you add more actions.

model_metadata.json (~/deepracer-for-dummies/docker/volumes/minio/bucket/custom_files) - gedit

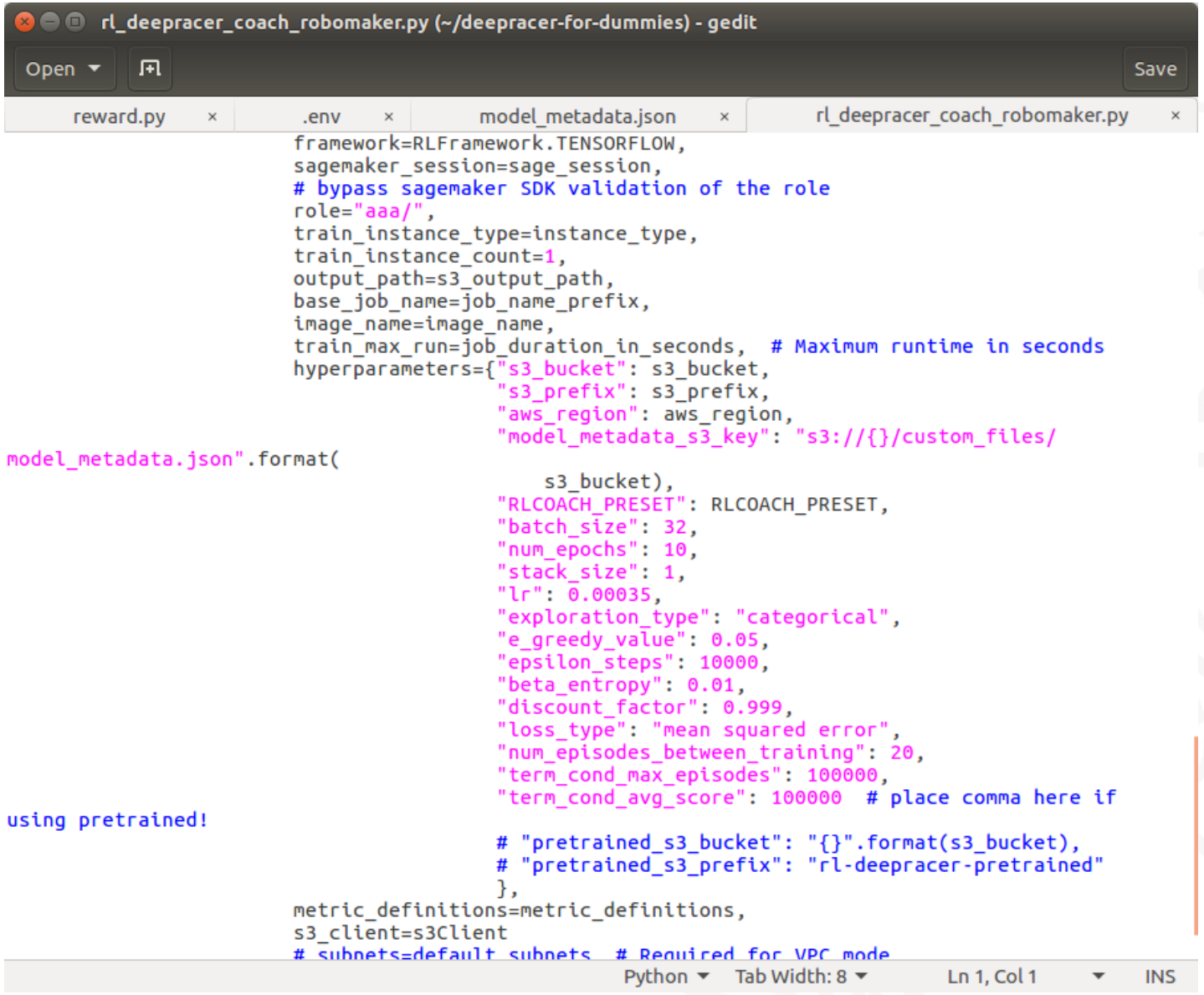
Open Save

reward.py | .env | model_metadata.json

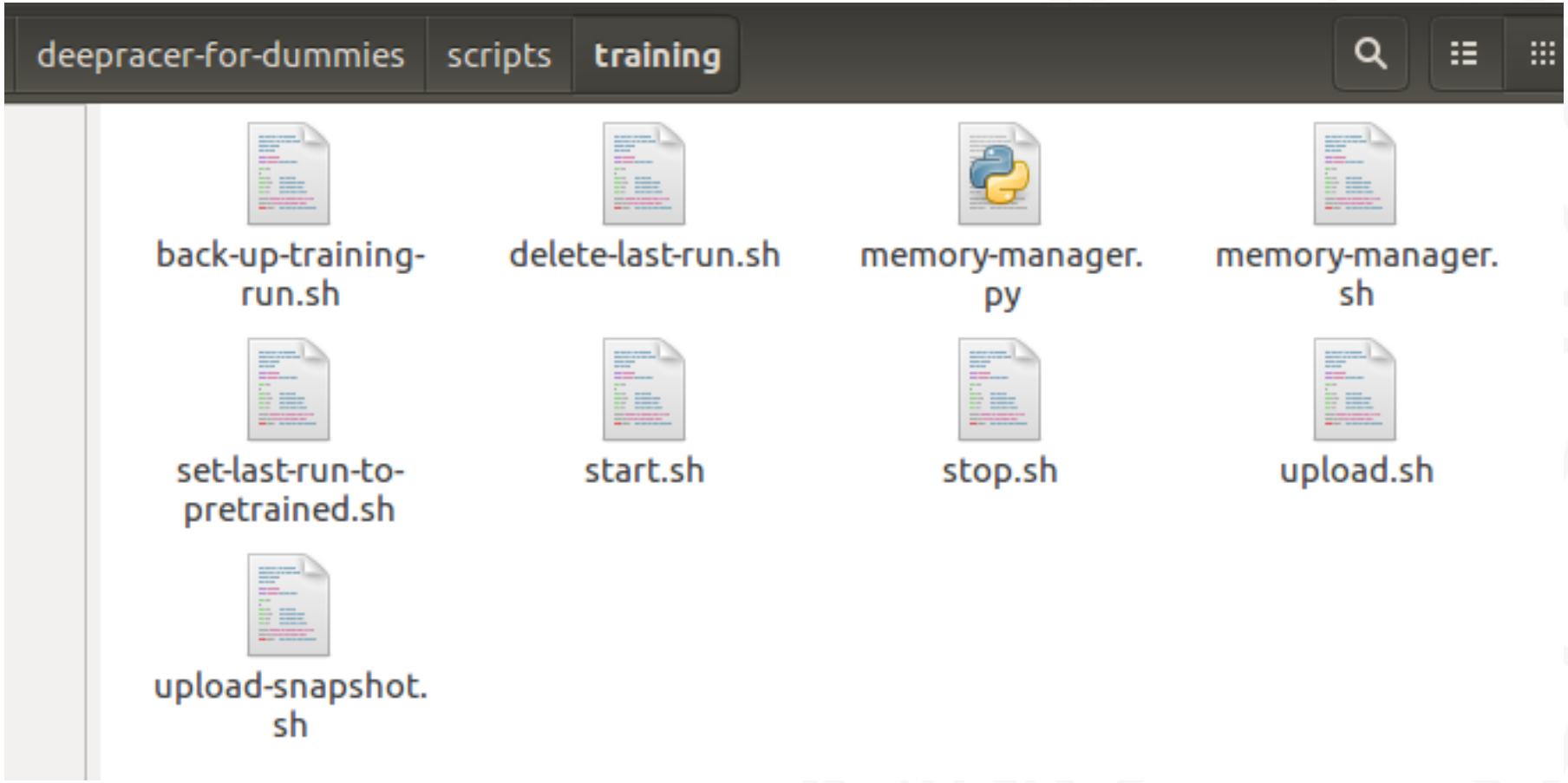
```
{
    "action_space": [
        {
            "steering_angle": -30,
            "speed": 0.4,
            "index": 0
        },
        {
            "steering_angle": -30,
            "speed": 0.8,
            "index": 1
        },
        {
            "steering_angle": -15,
            "speed": 0.4,
            "index": 2
        },
        {
            "steering_angle": -15,
            "speed": 0.8,
            "index": 3
        },
        {
            "steering_angle": 0,
            "speed": 0.4,
            "index": 4
        },
        {
            "steering_angle": 0,
            "speed": 0.8,
            "index": 5
        },
        {
            "steering_angle": 15,
            "speed": 0.4,
            "index": 6
        }
```

JSON Tab Width: 8 Ln 1, Col 1 INS

Adjust the hyperparameters in the deepracer-for-dummies/rl_deepracer_coach_robomaker.py file.

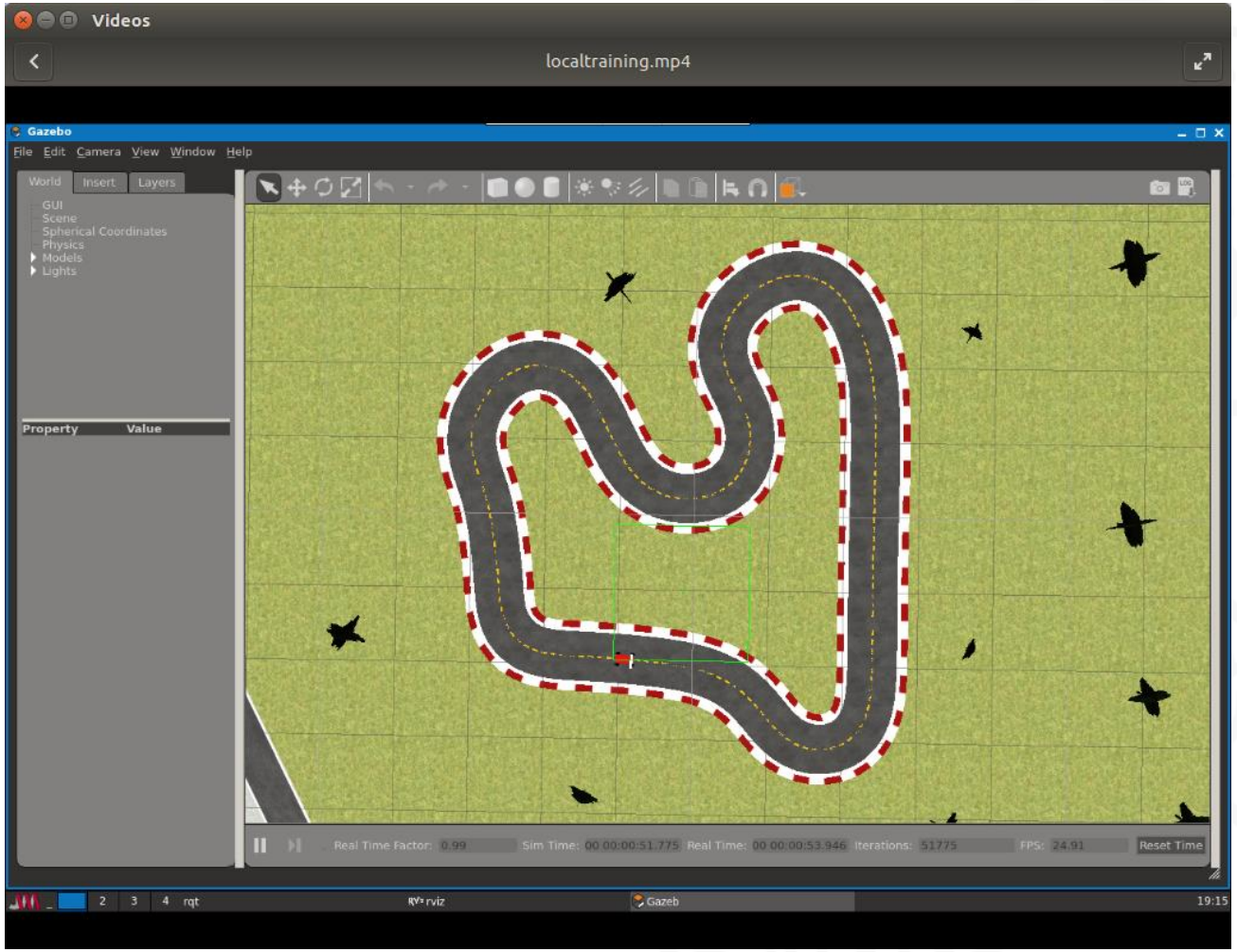


After edit the previously files, we can start training by run the script start.sh. Run stop.sh to stop the training and delete-last-run.sh to clean the space before next run.



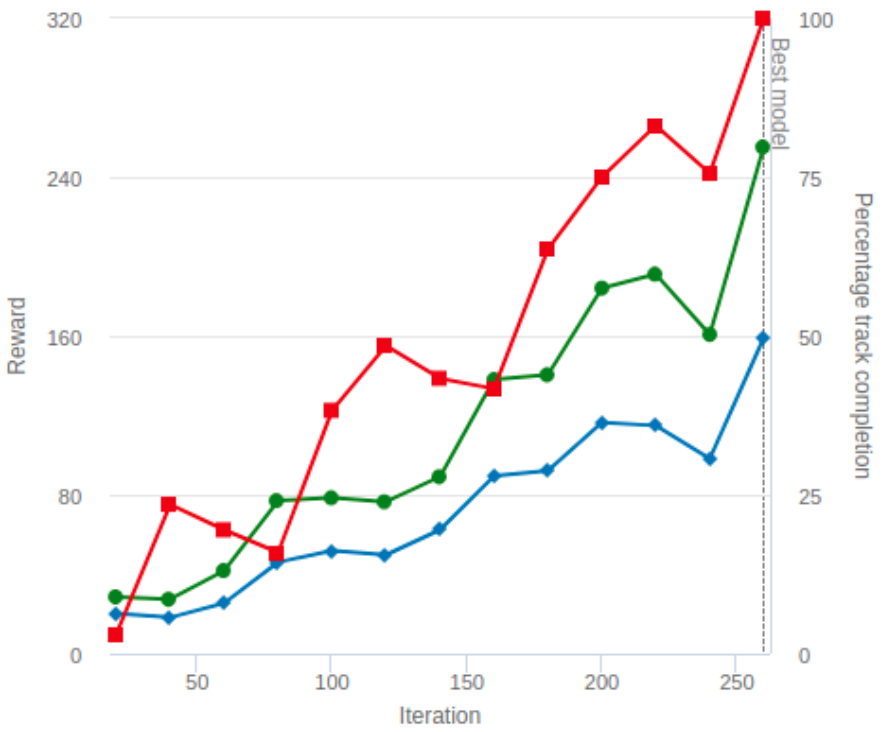
Local training using vncviewer for visualization

The track in the image is China_track map



Change reward function

Reward graph Info

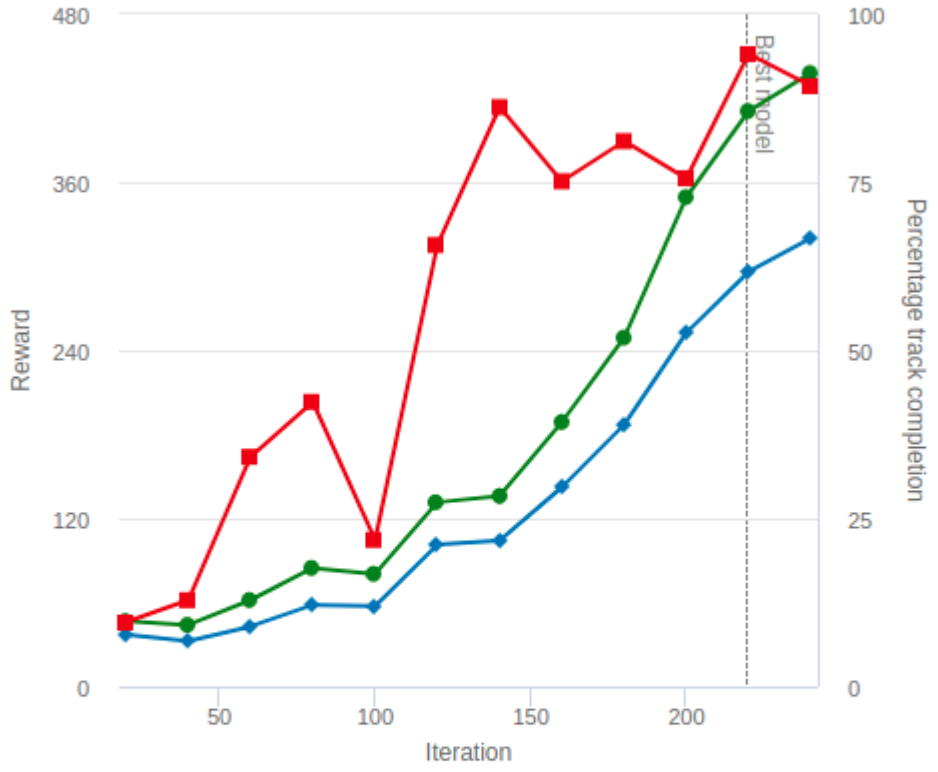


Evaluation results

Trial	Time	Trial results (% track completed)	Status
1	00:00:33.428	100%	Lap complete
2	00:00:31.085	100%	Lap complete
3	00:00:31.228	100%	Lap complete

Default reward function which is focus on follow the center line to get more rewards.

Reward graph Info

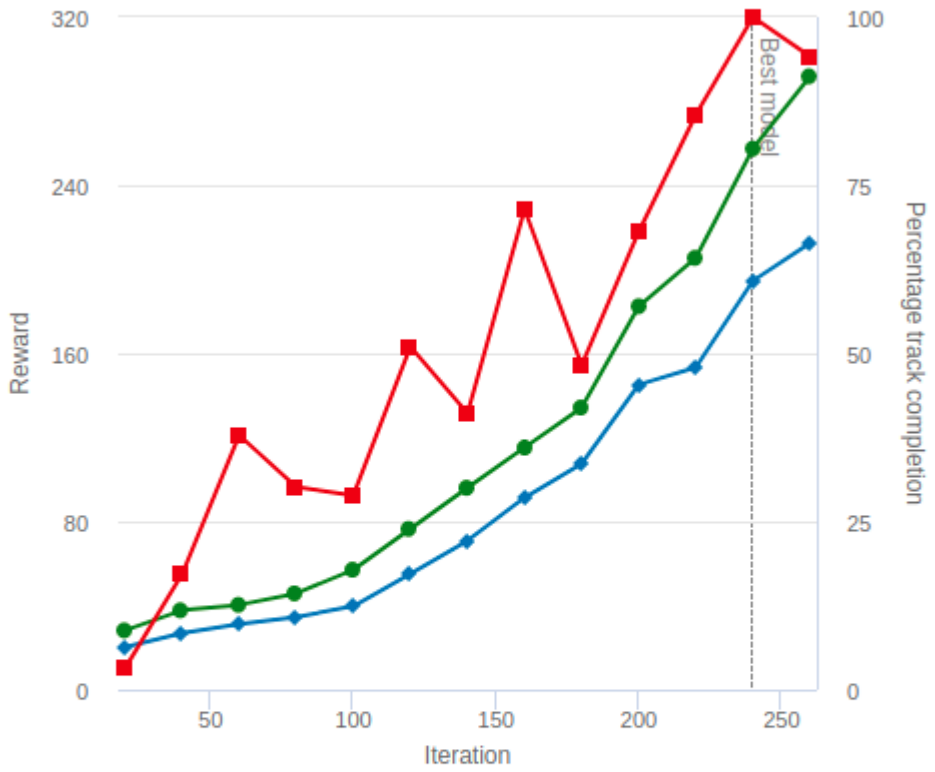


Evaluation results

Trial	Time	Trial results (% track completed)	Status
1	00:00:12.101	46%	Off track
2	00:00:25.681	100%	Lap complete
3	00:00:11.271	48%	Off track

I only use the all_wheels_ on_ track parameter to define the rewards. The rule is giving reward if there is no wheels go off the track.

Reward graph Info

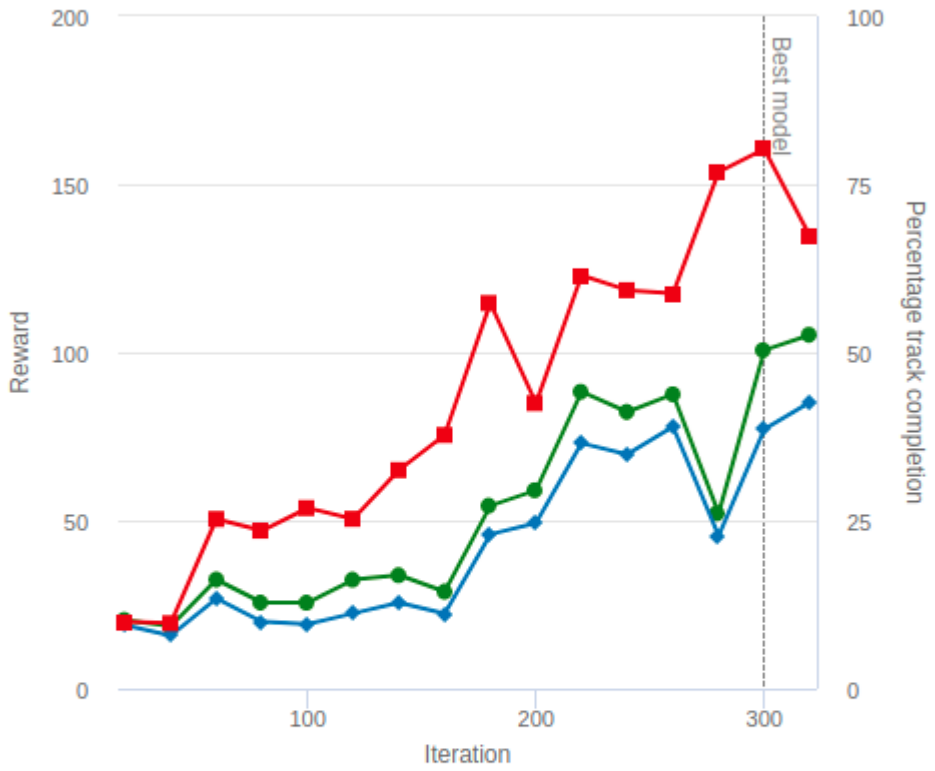


Evaluation results

Trial	Time	Trial results (% track completed)	Status
1	00:00:24.187	100%	Lap complete
2	00:00:17.232	71%	Off track
3	00:00:22.838	100%	Lap complete

This is based on the follow the center line function, and it add formula to penalize reward if the agent is steering too much.

Reward graph Info



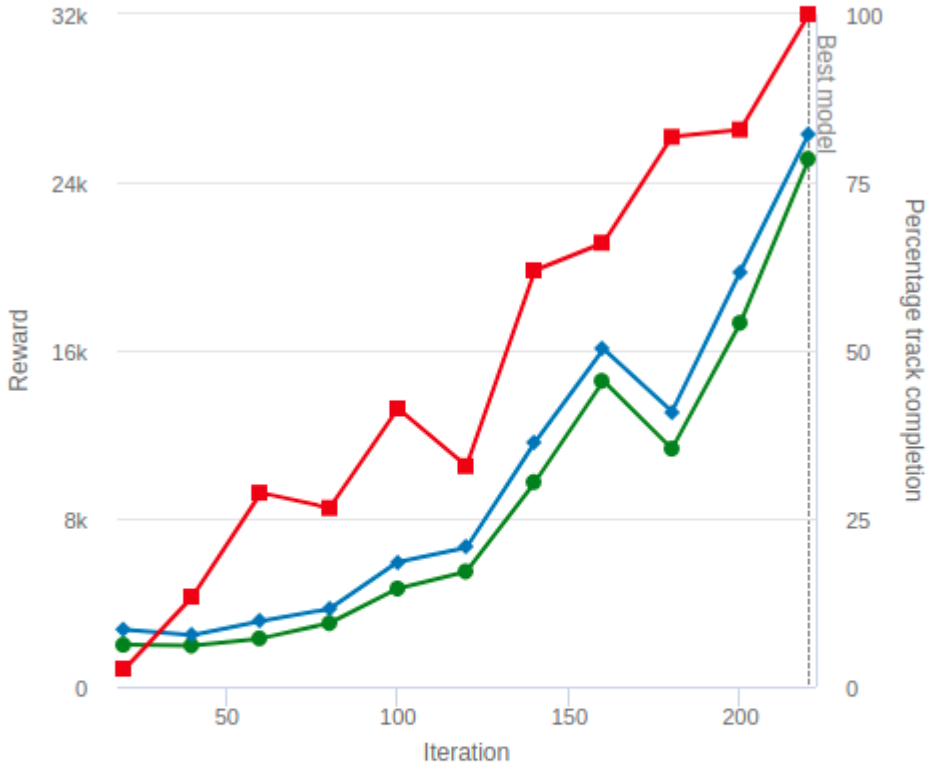
Evaluation results

Trial	Time	Trial results (% track completed)	Status
1	00:00:24.267	100%	Lap complete
2	00:00:23.426	100%	Lap complete
3	00:00:22.516	100%	Lap complete

This is based on the follow the center line function, and it add formula to give more reward if the agent using high speed to drive.

I use the parameter speed to increase the reward for every step.

Reward graph Info

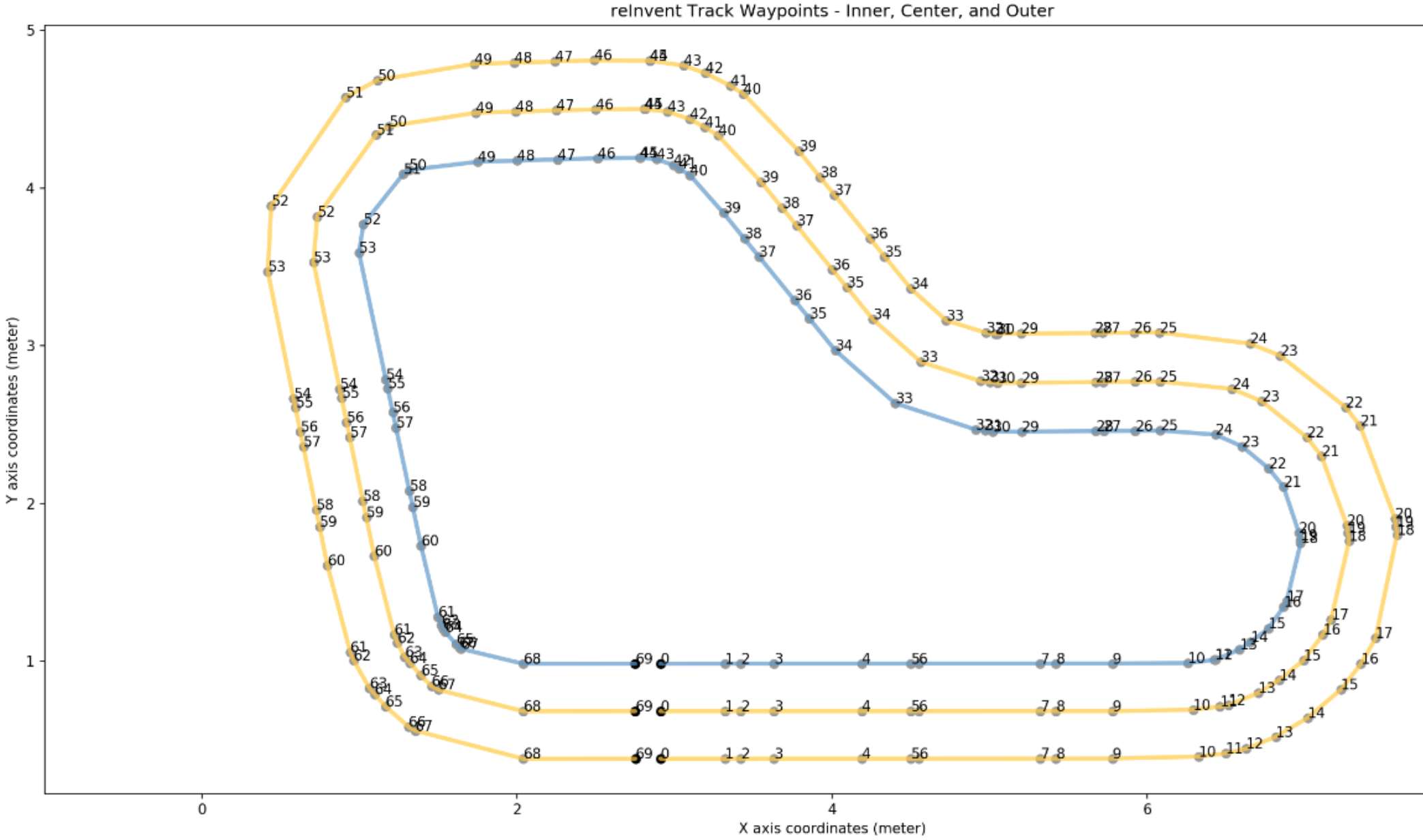


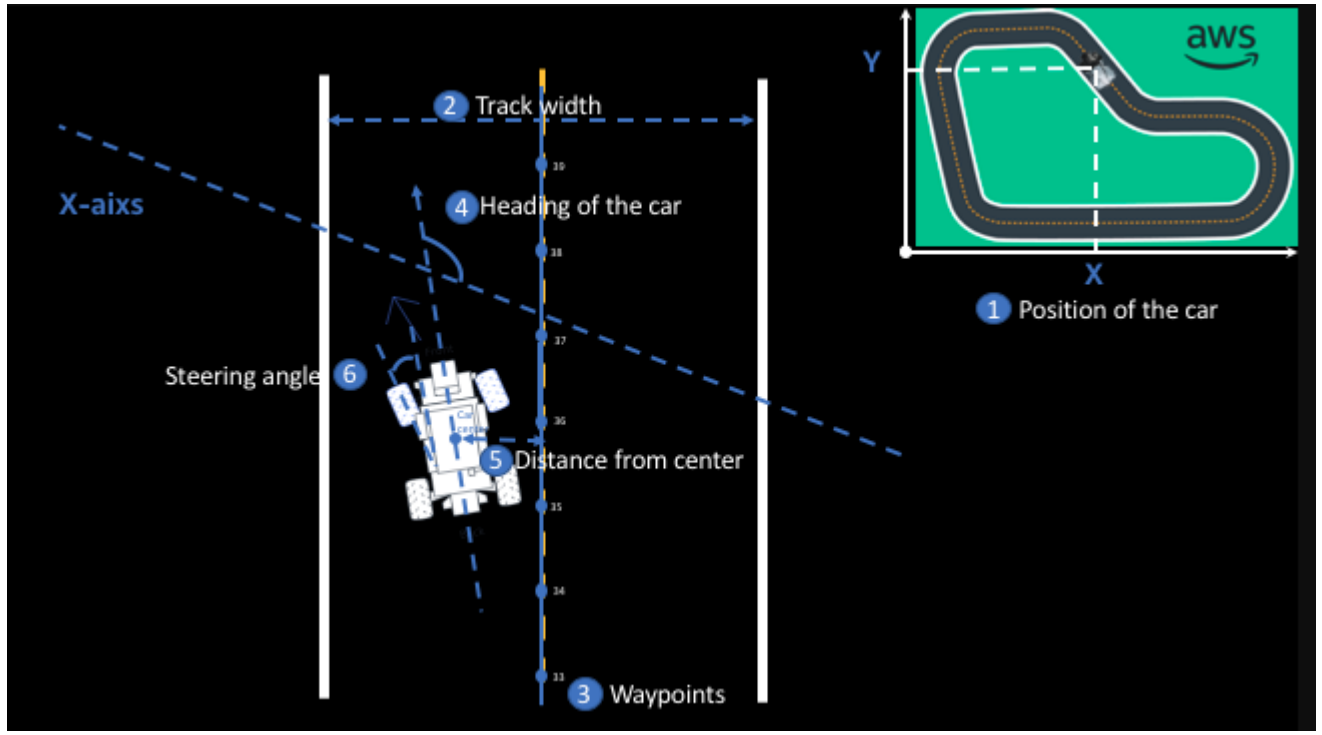
Evaluation results

Trial	Time	Trial results (% track completed)	Status
1	00:00:09.370	35%	Off track
2	00:00:28.909	100%	Lap complete
3	00:00:21.626	74%	Off track

This reward function is different from the previously reward functions. I use the waypoints as the reference to give rewards.

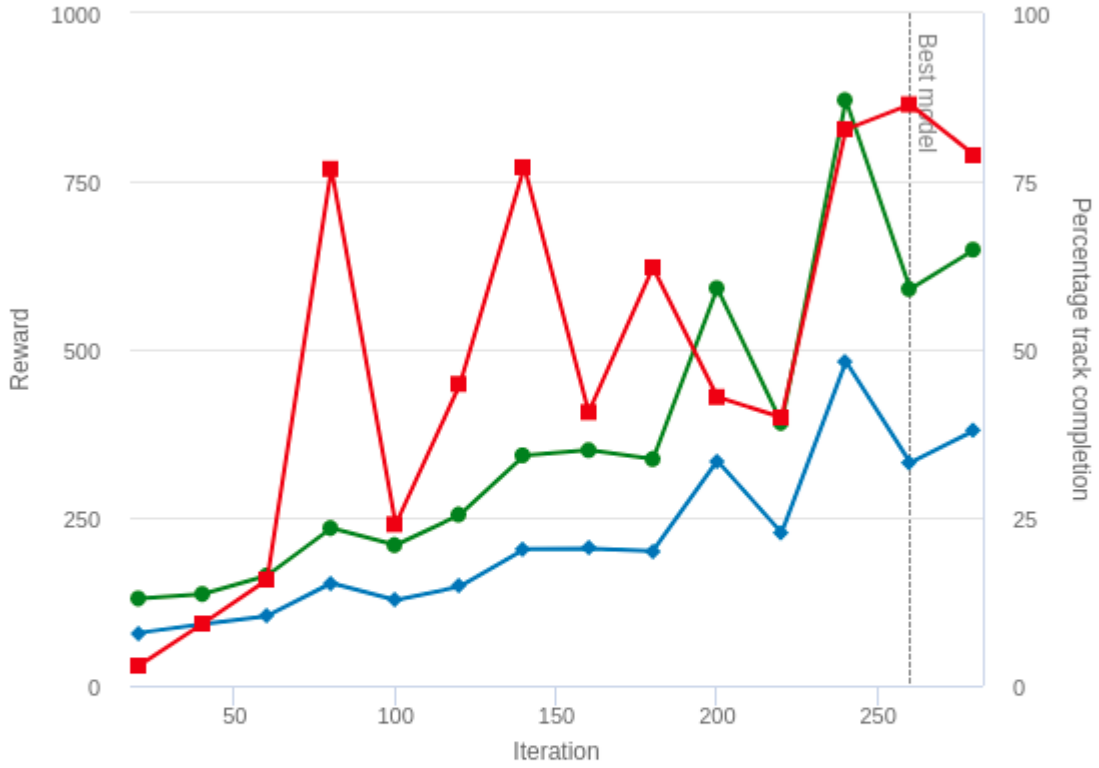
I firstly draw a shortest path on the track map, then I define where the agent locate to the center line, and use closest waypoints as reference.


reInvent Track Waypoints - Inner, Center, and Outer
Y axis coordinates (meter)
X axis coordinates (meter)
0
1
2
3
4
5
6
7
8



Variable Name	Type
all_wheels_on_track	Boolean
distance_from_center	Float
is_left_of_center	Boolean
speed	Float
steering_angle	Float
track_width	Float
closest_waypoints	Integer
progress	Float

Reward graph Info



Evaluation results

Trial	Time	Trial results (% track completed)	Status
1	00:00:26.232	100%	Lap complete
2	00:00:21.251	74%	Off track
3	00:00:09.964	35%	Off track

This is based on the waypoints following function. I add formula to give higher reward if the speed is fast and using less steps to finish the lap.

University at Buffalo The State University of New York

Thank you