

AWS DeepRacer-Basic

陳昀(Ingrid)



Lab

- Find the lab content here: [Link](#)
- Create a car
- Start training your time-trial model
- Submit your model to the Virtual Race

Your garage- Personalize

[AWS DeepRacer](#) > [Your garage](#) > Mod your vehicle

Step 1

Personalize

Step 2

Mod specifications

Personalize

Name

Give your vehicle a memorable moniker to help you remember its specifications.

Name your vehicle

The vehicle name must have 1-32 characters. Valid characters: A-Z, a-z, 0-9, _ (underscore) - (hyphen).

Cancel

Next

Your garage- Mod your own vehicle

- Camera
- Stereo camera
- LIDAR sensor

Mod your own vehicle

Mod specifications

Configure your vehicle with one or more sensors, choose a neural network topology, and customize the action space to meet your racing criteria. Customize your vehicle's appearance for personalized visualization in training.

Sensor modification

Swap sensors to improve your DeepRacer's racing performance

☒ Camera

Single-lens 120-degree field of view camera capturing at 15fps. The images are converted into greyscale before being fed to the neural network.

► Benefits of the front-facing camera

☐ Stereo camera

Composed of two single-lens cameras, stereo camera can generate depth information of the objects in front of the agent and thus be used to detect and avoid obstacles on the track. The cameras capture images with the same resolution and frequency. Images from both cameras are converted into grey scale, stacked and then fed into the neural network.

► Benefits of the stereo camera

Add-on sensors

☐ LIDAR sensor

LIDAR is a light detection and ranging sensor. It scans its environment and provides inputs to the model to determine when to overtake another vehicle and beat it to the finish line. It provides continuous visibility of its surroundings and can see in all directions and always know its distances from objects or other vehicles on the track.



Cancel

Previous

Done

Your models- Create model

▼ Racing League

AWS Virtual Circuit

Community races New!

Your racer profile

▼ Reinforcement learning

Get started

Your models

Your garage

▼ Resources

About the league [↗](#)

Schedules & standings [↗](#)

Rules & prizes [↗](#)

Developer guide [↗](#)

Tips & tricks [↗](#)

Forum [↗](#)

Community Slack channel [↗](#)

Buy AWS DeepRacer [↗](#)

AWS DeepRacer > Your models

New: Customize your action space for more control over your model's performance. [Learn more](#) [↗](#)

×

Models (11)

Import model

Actions ▼

Create model

Q Search models

< 1 2 > ⚙

	Name ▼	Description ▼	Status ▼	Algorithm ▼	Sensors ▼	Creation time ▼
<input type="radio"/>	reinvent2018-closetwaypoints		Ready	PPO	Camera	Fri, 15 Oct 2021 03:46:33 GMT
<input type="radio"/>	reinvent2018-gustav-test-clone		Ready	PPO	Camera	Fri, 15 Oct 2021 01:57:52 GMT
<input type="radio"/>	reinvent2018-gustav-test		Ready	PPO	Camera	Thu, 14 Oct 2021 17:03:20 GMT
<input type="radio"/>	reinvent2018-dataspring-test		Ready	PPO	Camera	Thu, 14 Oct 2021 16:54:55 GMT
<input type="radio"/>	reinvent2018-gustav-test-clone		Ready	PPO	Camera	Thu, 14 Oct 2021 16:54:55 GMT

Your models- Specify the model name and environment

- Model name

Training details

Model name

TopModel-re:InventTrack

The model name must be unique and can have up to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen). No spaces or underscores.

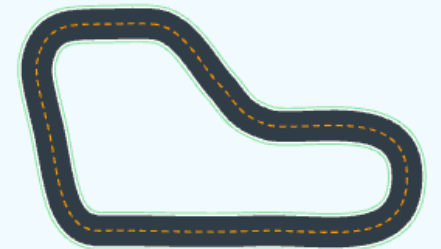
Training job description - optional

Log details for quick reference

The model description can have up to 255 characters.

- re:Invent 2018
Inspired by Monza, re:Invent 2018 was the first Championship Cup track. This short, classic speedway remains a perennial rookie favorite.

Length: 17.6 m (57.97')
Width: 76 cm (30")



Your models- Choose race type and training algorithm

- Race type



- Training algorithm and hyperparameters- Choose **PPO**

Training algorithm and hyperparameters [Info](#)

☒ **PPO**
A state-of-the-art policy gradient algorithm which uses two neural networks during training – a policy network and a value network.

☐ **SAC**
Not limiting itself to seeking only the maximum of lifetime rewards, this algorithm embraces exploration, incentivizing entropy in its pursuit of optimal policy.

► Hyperparameters

Cancel

Previous

Next

Your models- Define action space

- Select action space- **Discrete action space**
- Define discrete action space

Define discrete action space [Info](#)

Steering angle

The steering angle determines to what degree the front wheels of your agent can turn. Steering angle granularity describes the increments between angles. Choose higher numbers for smoother actions. Higher numbers also expand the action space and thus increase training time.

Pro tip: Too high a steering angle can empower the agent to make unnecessarily excessive turns and can cause zig-zagging.

Steering angle granularity

5 ▼

Maximum steering angle

30

degrees

Max values are between 1 and 30.

Speed

The speed determines how fast your agent can drive. For the agent to be able to drive faster, set a higher speed. On a given track, you must balance the desire for speed against the concern for keeping the agent on the track while it maneuvers curves at a high speed.

Pro tip: The higher the speed limit and more actions, the vehicle has a better chance of driving faster, but the model may take longer to converge.

Speed granularity

2 ▼

Maximum speed

2

m/s

Select values between 0.1 and 4.

Your models- Choose vehicle

- Choose **The Original DeepRacer**



Your models- Customize reward function

- Use Reward function examples to run your first model

Code editor

Reward function examples

Reset

Validate

```
1 def reward_function(params):
2     """
3     Example of rewarding the agent to follow center line
4     """
5
6     # Read input parameters
7     track_width = params['track_width']
8     distance_from_center = params['distance_from_center']
9
10    # Calculate 3 markers that are at varying distances away from the center line
11    marker_1 = 0.1 * track_width
12    marker_2 = 0.25 * track_width
13    marker_3 = 0.5 * track_width
14
15    # Give higher reward if the car is closer to center line and vice versa
16    if distance_from_center <= marker_1:
17        reward = 1.0
18    elif distance_from_center <= marker_2:
19        reward = 0.5
20    elif distance_from_center <= marker_3:
21        reward = 0.1
22    else:
23        reward = 1e-3 # likely crashed/ close to off track
24
25    return float(reward)
```

Your models- Customize reward function

- **Time trial - follow the center line (Default)**

This example determines how far away the agent is from the center line and gives higher reward if it is closer to the center of the track. It will incentivize the agent to closely follow the center line.

- **Time trial - stay inside the two borders**

This example simply gives high rewards if the agent stays inside the borders and lets the agent figure out what is the best path to finish a lap. It is easy to program and understand, but will be likely to take longer time to converge.

- **Time trial - prevent zig-zag**

This example incentivizes the agent to follow the center line but penalizes with lower reward if it steers too much, which will help prevent zig-zag behavior. The agent will learn to drive smoothly in the simulator and likely display the same behavior when deployed in the physical vehicle.

Your models- Customize reward function

- **Stop conditions**

Stop conditions [Info](#)

Set the conditions for your training job to stop. To avoid run-away jobs, you can limit the length of a job to within a maximum time period (**Maximum time**).

The training will stop when the specified criteria is met. When your model has stopped training, you will be able to clone your model to start training again using new parameters.

Maximum time

Maximum time must be between 5 and 1440 minutes.

AWS DeepRacer Reward Function Examples ([Link](#))

Example 1: Follow the Center Line in Time Trials

This example determines how far away the agent is from the center line, and gives higher reward if it is closer to the center of the track, encouraging the agent to closely follow the center line.

```
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 increasingly further 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  
  
    return reward
```

Example 2: Stay Inside the Two Borders in Time Trials

This example simply gives high rewards if the agent stays inside the borders, and let the agent figure out what is the best path to finish a lap. It is easy to program and understand, but likely takes longer to converge.

```
def reward_function(params):  
    '''  
    Example of rewarding the agent to stay inside the two borders of the track  
    '''  
  
    # Read input parameters  
    all_wheels_on_track = params['all_wheels_on_track']  
    distance_from_center = params['distance_from_center']  
    track_width = params['track_width']  
  
    # Give a very low reward by default  
    reward = 1e-3  
  
    # Give a high reward if no wheels go off the track and  
    # the car is somewhere in between the track borders  
    if all_wheels_on_track and (0.5*track_width - distance_from_center) >= 0.05:  
        reward = 1.0  
  
    # Always return a float value  
    return reward
```



Example 3: Prevent Zig-Zag in Time Trials

This example incentivizes the agent to follow the center line but penalizes with lower reward if it steers too much, which helps prevent zig-zag behavior. The agent learns to drive smoothly in the simulator and likely keeps the same behavior when deployed in the physical vehicle.

```
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']  
    abs_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 = 1.0  
    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  
  
    # Steering penalty threshold, change the number based on your action space setting  
    ABS_STEERING_THRESHOLD = 15  
  
    # Penalize reward if the car is steering too much  
    if abs_steering > ABS_STEERING_THRESHOLD:  
        reward *= 0.8  
  
    return float(reward)
```